NTNU – Trondheim
Norwegian University of
Science and Technology

Robotic Maintenance and ROS

Vegard Stjerna Lindrup

Submission date: June 2016
Responsible professor: Tor Engebret Onshus
Supervisor:

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Title: Robotic Maintenance and ROS

Student: Vegard Stjerna Lindrup

Problem description:

The goal of this thesis is to identify how an autonomous and remotely operated robot can utilize visual sensors. This project is a continuation of several projects on the topic of robotic maintenance. Therefore, the sensor applications should focus on maintenance tasks and movement of the mobile robot. The thesis should build upon the preceding project on this topic, which looked at depth perception for obstruction detection. To complete this goal, the student should:

1. Gain knowledge on theory, implementations and solutions for robotic maintenance and mobile autonomous robots that are relevant for this topic.
2. Present a set of development tools and hardware that are suitable in light of the goal above.
3. Propose a test platform for demonstrating some selected tasks performed by using vision based sensors. This platform may be based the existing mobile platform from the preceding project, either in its original or a modified form.
4. Develop solutions to at least one task that an autonomous maintenance robot could face, e.g. pick up an object, move to dock, or visual inspection. The solution must use a vision-based sensor to solve the task.
5. Assess the performance of the solution, and suggest improvements for the future.

Responsible professor: Tor Engebret Onshus

Abstract

Robotic maintenance has been a topic in several master's theses and specialization projects at the Department of Engineering Cybernetics (ITK) at NTNU over many years. This thesis continues on the same topic, with special focus on camera-based mapping and navigation in conjunction with automated maintenance, and automated maintenance in general. The objective of this thesis is to implement one or more functionalities based on camera-based sensors in a mobile autonomous robot. This is accomplished by acquiring knowledge of existing solutions and future requirements within automated maintenance.

A mobile robot prototype has been configured to run ROS (Robot Operating System), a middleware framework that is suited to the development of robotic systems. The system uses RTAB-Map (Real-Time Appearance Based Mapping) to survey the surroundings and a built navigation stack in ROS to navigate autonomously against easy targets in the map. The method uses a Kinect for Xbox 360 as the main sensor and a 2D laser scanner to the surveying and odometry.

It is also developed functional concepts for two support functions, an Android application for remote control over Bluetooth and a remote central (OCS) developed in Qt. Remote Central is a skeletal implementation that is able to remotely control the robot via WiFi, as well as to display video from the robot's camera.

Test results, obtained from both live and simulated trials, indicate that the robot is able to form 3D and 2D map of the surroundings. The method has weaknesses that are related to the ability to find visual features. Laser Based odometry can be tricked when the environment is changing, and when there are few unique features. Further testing has demonstrated that the robot can navigate autonomously, but there is still room for improvement. Better results can be achieved with a new movable platform and further tuning of the system.

In conclusion, ROS works well as a development tools for robots, and the current system is suitable for further development. RTAB-Maps suitability for use on an industrial installation is still uncertain and requires further testing.

Sammendrag

Robotisert vedlikehold har vært et tema i flere masteroppgaver og fordypningsprosjekter ved Institutt for teknisk kybernetikk (ITK) - NTNU over mange år. Denne oppgaven viderefører temaet, og ser nærmere på kamerabasert kartlegging og navigasjon i forbindelse med robotisert vedlikehold, samt robotisert vedlikehold generelt. Målet med oppgaven er å implementere én eller flere funksjonaliteter basert på kamerabaserte sensorer i en mobil autonom robot. Dette gjøres ved å skaffe kunnskap om eksisterende løsninger og fremtidige behov innen robotisert vedlikehold.

En mobil robot prototype er blitt konfigurert til å kjøre ROS (Robot Operating System), et mellomvare rammeverk som er velegnet til utvikling av robotsystemer. Systemet benytter RTAB-Map (Real-Time Appearance Based Mapping) til å kartlegge omgivelsene og en innebygget navigasjonsstack i ROS for å navigere autonomt mot enkle mål i kartet. Metoden benytter en Kinect for Xbox 360 som hovensensor, og en 2D laserskanner til både kartlegging og odometri.

Det er i tillegg utviklet fungerende konsepter for to støttefunksjoner, en Android applikasjon for fjernstyring over Bluetooth og en fjernstyringssentral (OCS) utviklet i Qt. Fjernstyringssentralen er en skjelettimplementasjon som er i stand til å fjernstyre roboten via Wifi, samt å vise video fra robotens kamera.

Testresultatene, som er inhentet fra simuleringer og testing av selve roboten, viser at roboten er i stand til å danne 3D- og 2D-kart av omgivelsene. Metoden har svakheter som er knyttet til evnen til å finne visuelle kjennemerker. Laserbasert odometri kan lures når omgivelsene er i endring, og når det er få unike kjennemerker. Videre testing har demonstrert at roboten kan navigere autonomt, men det er fortsatt rom for forbedringer. Bedre resultater kan oppnås med en ny bevegelig plattform og videre tuning av systemet.

Den endelige konklusjonen er at ROS fungerer godt som utviklingsverktøy for roboter, og at det nåværende systemet er egnet for videre utvikling. RTAB-Maps egnethet til bruk på en industriell installasjon er fremdeles usikkert, og krever videre testing.

Preface

This thesis concludes my time as a master's student at NTNU. The last five years have been brimming with challenges and learning experiences, but I believe that the last two years have been the most rewarding by far. Both this project, and last years specialization project has fueled my interest in computer vision and robotics. Another reason for this interest is probably also due to readily available libraries and frameworks such as ROS, Qt and OpenCV.

A challenging part of the project work this semester was the openness of the problem description. Looking for, and finding tools consumed a lot of time in the initial weeks. Some ideas turned out to be blind alleys or associated with and unacceptable level of uncertainty. On the other hand, this uncertainty made the project more interesting, and it could be that I never would have come around to use ROS without it.

To students who are considering to continue working on this project, I would recommend that they get a solid background in use of ROS before taking on this topic, perhaps by taking part in the subject TTK8, taught by Amund Skavhaug. I sincerely hope the information provided in the appendix and on the digital attachments will be of value during further development.

Vegard Stjerna Lindrup

Monday 13$^{\text{th}}$ June, 2016

Acknowledgments

This thesis, and the obtained project results would not have been possible without help from student colleagues, friends and support from the Department of Engineering Cybernetics.

I would first like to thank my project supervisor, Professor Tor Onshus of the Department of Engineering Cybernetics at NTNU, for allowing me to work on such an open and interesting topic, and for providing valuable advice and guidance through the two last semesters. He has been quick to respond to problems with the project, and gave me a much needed sense of urgency when the project was lagging far behind schedule in march.

Among my student colleagues, Eirik Wold Solnør, Vegard Blomseth Johnsen and Henrik Rudi Haave have been particularly helpful during testing sessions and for video documentation. Over the last two semesters, Ole Magnus Siqveland and I have used the same robot platform for our projects. Through good collaboration, we found a hardware setup that worked for both of us; a new shelf structure with compartments for the various hardware components.

I would like to thank the foreman, Terje Haugen, and apprentice Daniel Bogen at the mechanical workshop for building the new compartments and frames for the robot used in this thesis. Many thanks goes to the employees at the electronics workshop for allowing me to borrow tools and equipment, and providing some hints and tips.

I am very grateful to my parents, my sister, Jon Vedum and Andrea Myklebust for their support during the years at NTNU. Thank you!

Sincerely,
Vegard Stjerna Lindrup

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

AIV Autonomous Inspection Vehicle

CLM Concurrent Localization and Mapping

CP Cathodic Protection

DRC DARPA Robotics Challenge

DWA Dynamic Window Approach

EKF Extended Kalman Filter

EMC Electromagnetic compatibility

Fraunhofer IPA Fraunhofer Institute for Manufacturing Engineering and Automation

GUI Graphical User Interface

HMI Human-Machine Interaction

HSE Health, Safety and Environment

IFR International Federation of Robotics

IO Integrated Operations

ISS International Space Station

ITK Department of Engineering Cybernetics

LIDAR LIght Detection And Ranging

LTM Long Term Memory

MAR Mobile Autonomous Robot

MIMROex Mobile Inspection and Monitoring Robot, experimental

MIT Massachusetts Institute of Technology

NCS Norwegian Continental Shelf

NDT Non-destructive Testing

NUI Natural user interface

OCS Operator Control Station

O&G Oil & Gas

OpenCV Open Source Computer Vision Library

ORB Oriented FAST and Rotated BRIEF

PCL Point Cloud Library

PR Personal Robot

PWM Pulse Width Modulation

RBI Risk Based Inspection

ROS Robot Operating System

ROV Remotely Operated Vehicle

RPAS Remotely Piloted Aerial System

RTAB-Map Real-Time Appearance-Based Mapping

SDF Simulation Description Format

SIFT Scale-invariant feature transform

SIM Structural Integrity Management

SLAM Simultaneous Localization And Mapping

SSD Solid State Drive

STM Short Term Memory

SURF Speeded Up Robust Features

STAIR Stanford AI Robot

TLA Three Letter Acronym

UAV Unmanned Aerial Vehicle

URDF Unified Robot Description Format

UUID Universally Unique Identifier

VR Virtual Reality

WM Working Memory

Chapter 1
Introduction

1.1 About the Project

This thesis presents and documents this author's master's project, **TTK4900**, on robotic maintenance which was carried out at the Department of Engineering Cyberentics (ITK) in the spring of 2016. **TTK4900** is worth 30 credits (studiepoeng), and the project duration is set to 22 full-time weeks with the possibility of extension in case of a valid reason. The work presented in this thesis is carried out as an independent effort, which is supervised by Professor Tor Onshus through regular status meetings.

1.1.1 The Project Proposal - Mobile Autonomous Robot

The robot system that was used in this project has been developed over the course of many preceding master and specialization projects. The long term goal of these projects is to develop mobile autonomous robot concepts for maintenance and inspection on topside offshore installations. The topic of this thesis is based on the project proposal which is given by Professor Tor Onshus at Department of Engineering Cybernetics (ITK). A description of this proposal[1], suggests some possible applications for such a robot:

- The robot could serve in a supporting role as a part of Integrated Operations (IO).
- It can also be used to prepare a normally unmanned topside offshore installation before the arrival of a maintenance crew, by performing safety checks and preparing the helicopter landing pad.
- Allow personnel to perform remote inspection and maintenance through telepresence.

[1] http://folk.ntnu.no/onshus/Oppgaver.htm

– In combination with Virtual Reality (VR), the robot could be used for training purposes.

1.2 Preceding Projects

Telepresence and Robotic Arm

The system in its current form is built around a robot manipulator arm, SCORBOT-ER4u. Kristian Saxrud Bekken focused on improving previous work on the system, which was done as early as 2005[Bek10]. Bekken's work comprise telepresence through a stereo video transmission, a collision avoidance system for the robot arm and an improved HMI implementation.

Building the Mobile Platform

During the spring of 2013, Petter Aspunvik devoted his master's project to develop a mobile base for the robotic arm[Asp13]. Aspunvik's thesis has served as a user manual for many of the robot systems in the early stages of this project. The current motor control firmware used Aspunvik's implementation as a starting point.

Simultaneous Localization and Mapping

In parallel to Aspunvik's project, Mikael Berg developed a solution for Simultaneous Localization And Mapping (SLAM) and autonomous navigation for the same robot[Ber13]. His software is programmed in Google's Go language, and runs on Windows 7 within the pre-installed on-board computer. The resulting system successfully utilized Hector SLAM with a LIDAR and odometry from two encoder wheels for 2D navigation and SLAM. Berg considered to create a solution based on ROS which requires a Linux platform. In the end, he opted to target the Windows platform as this apparently is the only operating system which is compatible with the robotic arm. The "future work"-section in Berg's thesis suggests improvements in the form of 3D obstruction detection, because the LIDAR is limited to detection in a plane. He also mentions object recognition and dynamic re-planning as possible extensions.

Last Year's Specialization Project

This author's specialization project[Lin15] presented an obstruction detector based on two unsynchronized IP-cameras and a stereo matching algorithm in Open Source Computer Vision Library (OpenCV). Because the cameras were unsynchronized, the system would become useless whenever there is relative motion between the robot and the surroundings. The obstruction detector lacked a critical feature: a floor filter to separate the ground from potential obstructions. The implementation presented

in this master's project is unrelated to the preceding specialization project, except for some useful c++ functions and ideas that are brought forward.

1.3 Implementation Overview

Deciding on a Goal

To meet objective 4 in the problem description, it was decided to focus on vision based navigation. A robot with the ability to build a map of the surroundings and relocate autonomously was considered to be a good starting point for further development of vision based solutions.

Limitations

During the implementation process, the focus has been on getting the system to work. Robustness, optimization and elegance has been abandoned in favor of the opportunity to increase the scope of the project.

Selecting Tools and Hardware

As a continuation of this author's specialization project, the robot was equipped with a 3D camera, a Kinect for XBOX 360 capable of perceiving depth images at a high frame rate ($30Hz$).

To use the Kinect, the initial plan was to utilize the Point Cloud Library (PCL) in combination with a SLAM method (e.g. Kintinous[Kine]). This approach came with a high degree of uncertainty that would reduce the project scope significantly, and increase the likelihood of an unsatisfactory result. The Robot Operating System (ROS), an open source robot software framework, came up as an alternative tool late in January.

The work and solutions presented in this thesis revolves around the process of integrating ROS with the mobile robot from [Asp13] and [Ber13]. Installing ROS on Ubuntu Linux is by far the easiest way to begin using the framework. For this reason, and to avoid interfering with another project on the same robot, it was decided that an additional computer running Linux should be fitted to the robot. The mobile robot from [Asp13] and [Ber13] was refitted to accommodate the Kinect and the second computer. The new robot platform configuration is shown in figure 1.2.

Two supporting tools were implemented in addition to the robot software: a simple concept of an OCS and a hand held remote control implemented on a smartphone. An overview of the complete system is shown in figure 1.1.

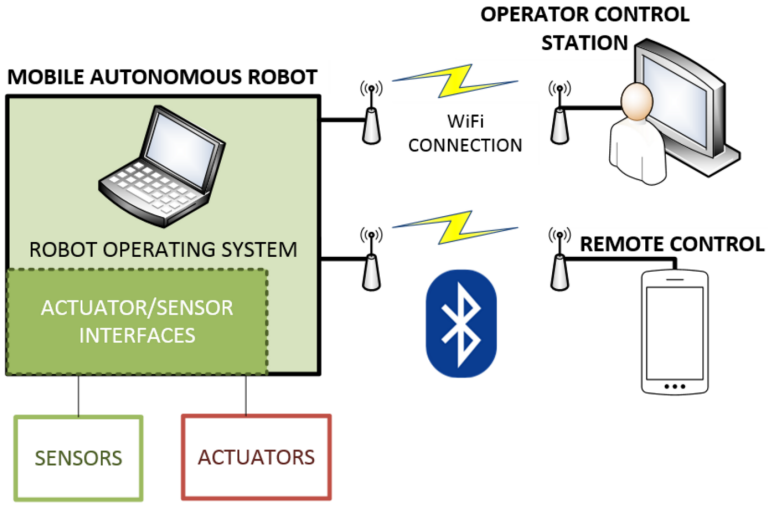
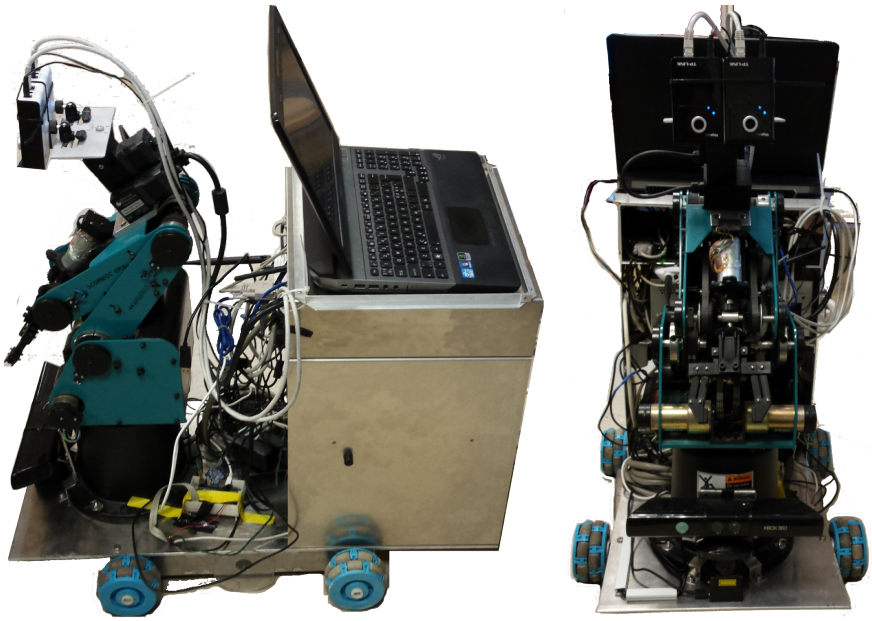


Figure 1.1: System Concept. An on-board computer using ROS to handle actuators and sensors. Remote operation is available through an OCS or a hand-held device with Bluetooth.



(a) Robot left side.

(b) Robot front.

Figure 1.2

Functionality

1. Simultaneous localization and mapping based on computer vision.
2. Mapping over multiple sessions.
3. Autonomous navigation to a simple goal.
4. 3D and 2D obstacle avoidance in navigation mode.
5. The robot can be controlled from the on-board keyboard.
6. The robot can be controlled by an Android Smartphone.
7. The robot can stream video to an URL.
8. The robot can receive velocity commands over WiFi.

1.4 Thesis Structure

Chapter 2 - Robotic Maintenance on Topside Offshore Platforms

Chapter 2 serves as motivation and background for the rest of the thesis. The chapter is to explain the state of automated maintenance is today, and to create a basis for comparison between field tested solutions and this prototype.

Chapter 3 - Background Theory

The next chapter introduces the background theory which is necessary to understand the implementations presented in chapter 4.

Chapter 4 - Implementation

Presents the implementations that were developed during the project. The structure of the robot system, both hardware and software, is documented.

Chapter 5 - Results

This chapter explains how the implementations were tested, and presents the ensuing test results.

Chapter 6 - Discussion

The discussion chapter contains an assessment of the results and implementations from chapters 4 and 5. The findings are discussed in light their fitness for robotic maintenance. The last section of this chapter

Chapter 7 - Conclusion

The concluding chapter will highlight how well the objectives in the initial problem description are covered. The final result and recommendations for future work is summed up in a final conclusion at the end of this chapter.

Chapter 2
Robotic Maintenance on Topside Offshore Platforms

2.1 Introduction

This project is a small step towards a larger long-term goal concerning robotic maintenance on topside offshore installations. This chapter puts the implementation described in chapter 4 into the context of the long-term goal.

Over the last decade, the Oil & Gas (O&G) industry has shown an increased interest in the potential benefits of automating the normal operation of remote offshore installations. Some satellite platforms, such as Sleipner B are already unmanned during normal operation, and the more recent Valemon platform is planned to become normally unmanned as well. It is now clear that robotics has several potential applications in process plants, and particularly in remote O&G installations. It is, however, difficult to predict how and to what extent robots will be applied in plant automation as other innovative solutions may arise[sta][sub][E24]. Current research on plant automation is mainly motivated by two factors[KMP15] [AS12]:

- **HSE** - Reduced risk exposure for personnel and environment.
- **Efficiency** - Accomplish more with less effort, resources and time. This means cost reduction by keeping downtime to a minimum with the least amount of effort.

An additional overarching driving factor is that O&G fields are becoming more difficult to reach. As O&G fields in shallow waters are depleted, production is moved to deeper waters. This complicates the extraction process and reduces the profit margin. A solution to this challenge is to increase efficiency through further automation.

This chapter provides a brief introduction to how topside maintenance is performed today, how these maintenance tasks could be robotized. The chapter is concluded with a brief discussion on how well modern robotics is suited for the task.

2.2 Robotizing Offhsore Maintenance

Recent research projects, concept labs and prototypes seem to focus on a solution where stationary or mobile robots serve in a supporting role in parallel to the dedicated process automation systems[AS12][KLT09][GP08][PBB11].

A feasibility study performed by researchers from Fraunhofer Institute for Manufacturing Engineering and Automation (Fraunhofer IPA)[PBB11] identified several topside production tasks and ranked them according to their resource demand. Based on the identified tasks, the study went on to describe a set of specific tasks, and then assess how easy or hard it would be to "robotize" these tasks. Table 2.1 associates a set of tasks with different robot categories, as well as how easy or hard the process of "robotizing" the activity is expected to be. The difficulty levels are described with the letters "A" through "D", where "A" is described as of the shelf robotics, which makes "robotizing" easy. Activities associated with the letter "D" on the other hand, cannot be be "robotized" with either current or near future technology even if doing so would be beneficial [PBB11]. Note that the paper in question is from 2011, and the difficulty of these tasks may have changed. This is particularly true in the domain of visual sensors, given the bloom of accessible 3D sensor technology and research over the last decade. Some further elaboration on inspection activities and environmental considerations is given in the next subsection.

The feasability study from [PBB11] suggests five tasks that will have a high impact on Health, Safety and Environment (HSE) and efficiency:

- Monitoring of gauges and meters
- (Visual) inspection of remote operated valves
- Acoustic inspection
- Inspection of equipment for leakage
- Maintenance of gas and fire sensors

Recovery scenarios is another area of application for offshore robots. As explained in [KMP15], a robot could be used to handle hazardous events that will lead to an evacuation of platform personnel. The robot could be a valuable tool in containing the hazardous situation quickly, thus reducing production downtime.

Robot Applications Including Categorization		
Category	Robot	Robot Task/Activity Description
B	Pipeline rigging robot	To autonomously load and offload pigs into pipelines.
C	Boat handling robot < 500 kg	To transfer personnel and loads below 500 kg to and from boats.
D	Boat handling robot > 500 kg	To transfer loads above 500 kg to and from boats.
B	Mobile universal service robot version 1	To perform "buddy" roles; carrying, holding, lifting, personal safety monitoring etc.
C	Mobile universal service robot version 2	To autonomously perform task not involving manipulation of the process or facilities.
D	Mobile universal service robot version 3	To autonomously perform tasks involving manipulation of the process or facilities.
D	Treatment/Inspection robot	To autonomously perform inspection/treatment(painting) tasks of structures/vessels or facilities.
A	Domestic service robot	To autonomously perform floor cleaning, catering, laundry handling, storage handling and logistics activities.

Table 2.1: Some tasks with potential to be "robotized", and the corresponding robot category[PBB11].

[AS12] from ABB suggests a stepwise approach toward robotization. The suggested approach is to break each concept and problem down into specific manageable tasks, before allowing the developed solutions to mature by exposing them to increasingly realistic test scenarios. Another path towards robotization corresponds to the difficulty assessments and classifications in table 2.1[GP08]. This roadmap starts with the bare necessities, e.g. tele-operation, ATEX certifications and safety. This will provide the foundation for developing inspection or surveillance robots, before the complexity can be increased by allowing robots to interact with and manipulate the processes. The end-goal is of course a fully autonomous maintenance system capable of operating the process itself[GP08].

2.2.1 Structural Maintenance and Environmental Considerations

Corrosion

Offshore installations are regularly, if not continuously, exposed to harsh weather conditions in the form of wind and seawater. Presence of seawater, either through direct contact or in the form of drops and vapor, forms a very corrosive environment. The offshore and marine environment is classified as the most corrosive environment in ISO 12944[ER12a]. It is essential to provide countermeasures to ensure safe and reliable operation over the lifetime of the installation. Common corrosion prevention methods are[ER12a]:

- Sacrificial Anodes.
- Cathodic Protection (CP) in the form of a DC-current.
- Protective coating.

In terms of maintenance, the sacrificial anodes can be subjected to periodic inspections and replacements, which could be done by a robot. CP can more easily be implemented with automated self tests, and should normally not require any inspections and maintenance[ER12a]. Application of protective coating should ideally be applied in the controlled environment of a workshop. If protective coating is to be applied at sea, one should strive to make the conditions as favourable as possible.

Structural Fatigue

Waves, wind, water currents and other forces subject offshore installations to structural stress. To keep the offshore installations from failing in these conditions, they may be subjected to a Risk Based Inspection (RBI) regime. In brief, RBI is a strategy where inspection and maintenance programs are developed based on which risk factors an installation is exposed to. In an automated maintenance program, an autonomous robot could perform inspections of the structure and generate reports based on risk factors such as[ER12b]:

Marine growth at sea level Marine growth will increase the diameter of supporting legs at sea level, thus increasing structural loads caused by waves, wind and water currents.

Corrosion Assess the seriousness of a corrosion attack through Non-destructive Testing (NDT).

Scour Scour around the platform legs could reduce a platforms ability to withstand structural loads. This is only applicable to non-floating installations.

A maintenance expert can then plan a maintenance campaign based on data from the robot in combination with knowledge on the platforms design, age and exposure to the environment.

2.2.2 Production-specific Hazards

O&G production has several inherent hazards, and an unwanted incident may have serious implications for HSE and production uptime. The Piper Alpha incident serves as a worst case example of the consequences of an explosive ignition of a hydrocarbon leak followed by an escalating fire. This section will briefly discuss some of the most significant hazards on an offshore oil and gas production plant.

Hydrocarbon leaks

Hydrocarbon leaks do occur on a regular basis. Over a four year period from 2006 to 2010, seven leaks larger than $1kg/s$ of either oil or gas/two-phase occurred in the Norwegian sector. No such leaks have ignited on the Norwegian Continental Shelf (NCS) since 1992. Of all the leaks which occurred in the same area, NCS, the majority was caused by human intervention[Vin14]. This could imply that a reliable robotic system may reduce the number of leaks.

Fire

Critical fire loads[1] on offshore facilities are usually caused by uncontrolled flow of hydrocarbons. The most serious of such releases is a blow-out. Risk reducing measures focus on four areas[Vin14]:

Leak prevention - Use equipment and assembly methods which minimize risk.

Leak detection - Fire & gas detection, emergency shut-down systems and blow-down systems.

Ignition prevention - Inspection and maintenance and Ex-approved equipment.

Escalation protection - Installation layout and sectioning. Fire and gas protection systems.

Explosions

Explosion protection is usually built into the equipment and structure. In [Vin14], there are no obvious ways a robot could provide additional explosion beyond e.g. leak detection, inspection and maintenance.

[1]Fire load can be defined as the amount of combustible material in a given area (Ref. https://en.wiktionary.org/wiki/fire_load).

2.2.3 Implications for Robot Design

A mobile robot operating on a normally unmanned platform in a harsh environment, implies that it is subjected to many of the same design philosophies that apply to subsea equipment.

Because of the risk of explosive atmospheres in an offshore production environment, an offshore robot operating under EU or EEA legislation will also be subject to the ATEX (ATmosphères EXplosibles) directive. Such a robot will most likely carry ignition sources such as batteries packed with energy and perhaps even welding equipment. A central ATEX requirement is to perform a risk assessment. As outlined by ATEX 2014-34-EU Guidelines[ATE], such a risk assessment is usually performed in four steps:

1. **Hazard identification** What can go wrong? Identify possible ignition sources, and the probability of explosive atmospheres.
2. **Risk estimation** Estimate the probability of an unwanted occurrence (e.g. an explosion), and the associated consequences.
3. **Risk evaluation** Evaluate the identified risk in context of acceptable risk, and decide if the design should be altered or if additional barriers should be installed. Barriers could either mitigate the consequences of an explosion, or reduce the possibility of ever having an explosion.
4. **Risk reduction option analysis** Identify possible risk reduction measures, e.g. barriers and design changes. A cost-benefit analysis can be performed in accordance with the ALARP-principle.

The on-board embedded computer hardware and software should be designed for robustness, fault tolerance and endurance. If a failure occurs, corrective actions will most likely be both difficult and expensive. Resistance to corrosion and toxic environments should also be taken into account.

Other design choices, e.g. the shape, size and equipment must fit the robots purpose. A source of guidelines for a mobile robot can draw upon case studies such as e.g. [GP08].

2.3 Robotic Maintenance Today

2.3.1 Trends and Potential

The typical pre-programmed assembly robots still dominate the robotic market. They are usually found in manufacturing plants and large scale production facilities[ifr],

e.g. the automotive industry, where they perform dull, tedious tasks much faster and with higher accuracy than people. A notable trend in modern robotics is increased human-robot collaboration[Bog16]. Many new robots are being built for the human workspace, both in terms of safety and collaborative functionality. This trend is a step along the way of moving robots out of the controlled environment of a factory floor, and into the real world where a high degree of autonomy is required.

A report by Metra Martech[GC11], a market research firm referenced to by International Federation of Robotics (IFR)[2], points to three areas with a high potential for robotic applications:

- Dangerous jobs, e.g. handling dangerous materials or work in high risk environments.
- Jobs that are economically infeasible in a high wage economy.
- Work which is impossible or highly inconvenient for humans, e.g. space exploration, subsea maintenance or assembly of heavy components.

All of these factors motivate the development of robots for autonomous robotic maintenance.

2.3.2 Subsea Maintenance and Inspection

Subsea maintenance is perhaps the field that have seen the greatest advancements in autonomous inspection and maintenance. As offshore installations are moved to the seabed, maintenance and inspection has become a significant challenge. This has resulted in a widespread use of Remotely Operated Vehicles (ROVs). Recent developments in other fields, e.g. computer vision, human-robot collaboration and machine learning, has resulted in new Autonomous Inspection Vehicles (AIVs) and Autonomous Underwater Vehicles (AUVs) capable of performing inspection and simple maintenance tasks

Figure 2.1: Subsea 7's AIV. This is the first commercial autonomous inspection vehicle for subsea operations [pre]

[2] http://www.ifr.org/robots-create-jobs/

autonomously[JWA+12][RCR+15]. A driving factor behind the transition from ROVs to AUVs is cost reduction through increased offshore campaign efficiency.

2.3.3 Disaster Response

Robots in disaster response, relief and recovery solve many of the same problems faced by maintenance robots. Disasters, such as the tsunami which struck Japan in 2011, proved that much work needs to be done, both in terms of technical capabilities and logistical issues related to deployment and response times. The tsunami resulted in three core meltdowns at the Fukushima Daiichi Nuclear Power plant[Ama15].

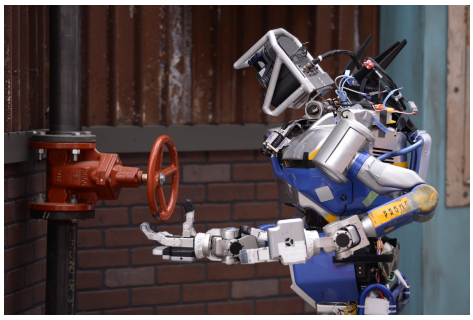


Figure 2.2: Team HRP2-Tokyo's robot turning a valve during DARPA Robotics Challenge 2015 (Image credits: DARPA Robotics Challenge)

Many of the robots which were deployed at the Fukushima Power Plant were already ageing, and the operators had to receive training before deployment, thus increasing the response time[KFO12]. A paper from Japan Atomic Energy Agency[KFO12] highlights how the lack of stakeholder involvement could have been the cause of long response times. The same paper points out that the robots were developed for the sake of development, and not with emergency response as the main purpose[KFO12].

DARPA Robotics Challenge (DRC)[DRC] was launched in response to the Fukushima disaster of 2011. The purpose of the competition is to accelerate innovation, research and development in robotics for disaster response in cases where humans cannot operate. Some of the tasks the competitors faces in 2015 include valve turning, traversing rubble and driving a vehicle through a course before egressing out of the vehicle.

2.3.4 Topside Offshore and Onshore Robotic Maintenance

Today, autonomous and teleoperated inspection and maintenance is usually only found at subsea installations. Topside installations on the other hand are still maintained and inspected manually, with some notable exceptions. Small Unmanned Aerial Vehicles (UAVs) or Remotely Piloted Aerial Systems (RPAS) have become commonplace and accessible to all over the last decade. There are currently RPAS

systems which are being used for visual inspection of inaccessible structural parts such as flare stacks or the exterior of oil rigs.

Some notable contributors to the field of robotic maintenance for O&G include ABB, Fraunhofer IPA, Sintef ICT[KLT09] and NREC at Carnegie Mellon University.

NRECs contribution, Sensabot, is a remotely operated inspection robot designed for harsh and remote environments[dep12]. It is not designed to be autonomous, but rather as a tool to move personnel from hazardous environments to safe remote control rooms. Sensabot mark II will be certified for zone 1 explosive environments. This year (2016), the plan is to test the robot on site at the Kashagan field in Kazakhstan[PSM+16].

Figure 2.3: An early version of the maintenance robot "Sensabot", developed by National Robotic Engineering Center (NREC) (Image credits: NREC)

Fraunhofer IPA[3] has developed a robot, called Mobile Inspection and Monitoring Robot, experimental (MIMROex). MIMROex has capabilities which are quite similar to the prototype used during the work on this thesis. MIMROex is equipped with a camera for visual inspections as well as microphones, vibration and sensors for fire and gas detection. It is also certifiable in accordance with the explosion protection standard IEC 60079[MIM]. Fraunhofer IPA has put great emphasis on field testing on actual offshore installations.

Both ABB and SINTEF ICT have developed lab facilities to test various concepts for robotic maintenance. Both facilities use non-mobile or semi mobile (gantries) robots which utilize a rich set of inspection and manipulation tools, as well as HMI equipment for remote operation and control. The two research communities differ in that ABB has tested their solutions in real environments, which subjects their solution to ATEX requirements and an extensive risk management regime[AS12]. SINTEF has a notable list of contributions to computer vision and 3D camera optics[4], and some of this research is geared towards inspection and maintenance. An example of this is a robot concept for replacing a battery in a wireless sensor by using 3D object detection[TSSO+10].

[3] http://www.ipa.fraunhofer.de/en.html

[4] http://www.sintef.no/en/information-and-communication-technology-ict/optical-measurement-systems-and-dataanalysis/

Another effort towards robotic maintenance is the ARGOS challenge (Autonomous Robot for Gas and Oil Sites). The purpose of the challenge is to promote innovation, understanding and awareness towards robotic maintenance of O&G sites in harsh environments[ARG][KMP15].

Chapter 3
Background Theory

3.1 Introduction

This chapter will provide the background theory which is necessary to understand the implementations described in chapter 4. Section 3.1.1 introduces some robot terminology and concepts. Section 3.2 provides a thorough introduction to ROS, which is the framework of choice in this implementation. Next, follows a section on sensors. More specifically the Kinect for Xbox 360 and a LIght Detection And Ranging (LIDAR), URG-04LX-UG01. The two final sections provide basic insight into SLAM and autonomous navigation in ROS respectively.

3.1.1 Brief Introduction to Robot Terminology

Joints and Links

A robot can be described by a set of rigid *links* connected to each other by *joints*. A link is described by a set of kinematic attributes based on its shape and mass. A joint between two links describe the freedom of movement between the coordinate systems, or frames, of each link. The link can also define the linear translation and rotations from parent frame to child frame. When a set of links and joints are put together, they will define the kinematic tree of the robot, i.e. how the robot and its components can move. Typical joint classes are:

Static	Transforms between links are constant.
Revolute	A rotary motion between the links - like a door hinge or a knee.
Prismatic	A linear motion between the links.
Continous	Unbounded rotary motion. Typically used for rotating wheels.

Mobile Bases

All ground based mobile bases can be separated into two main categories: holonomic or non-holonomic. In this thesis, a holonomic drive system will have three degrees of freedom, two of which are translational in the xy-plane, and the third accounts for rotation about the vertical z-axis. Holonomic drive robots will often be equipped with mechanum wheels or omniwheels arranged in a specific pattern. A non-holonomic drive will usually be constrained to forwards/backwards translation and a rotation about the vertical axis. Ackerman steering is the solution found in cars. Robots will often use a differential drive systems, which two large driven wheels, and at least one supporting wheel, for example a caster wheel.

For a navigation system, the major difference between a holonomic and non-holonomic is essentially that a holonomic drive is path independent, while a non-holonomic drive is path dependent. This implies that a holonomic drive can simply translate towards the goal state, while the non-holonomic drive must execute a sequence of rotations and translations to reach the goal state. The potential benefits of a holonomic drive becomes clear when considering the problem of parallel parking.

The mobile base used in this thesis is a hybrid between a skid steering drive, as in tracked vehicles, and a differential drive. The kinematics of the base is modeled as if it is a differential drive in section 4.6.2.

3.2 Robot Operating System (ROS)

3.2.1 Introduction

ROS is a collection of software libraries, tools and drivers intended for robot software development. A ROS installation can be tailored to meet the demands of a wide range of robots with varying complexity. ROS is usually installed in the form of an already built Debian-package. These packages are only compatible with a few versions of Ubuntu which are specified on the ROS homepage. When installed and configured, ROS will run on top of Linux, and can be perceived as and extention of Linux itself. Installing ROS from source is possible, but not recommended [ROSf].

Historic roots of ROS can be traced back to Stanford University at the beginning of the 2000s. At Stanford, several robotics software frameworks, including Stanford AI Robot (STAIR) and the Personal Robot (PR) program, were created to provide dynamic, flexible and well tested foundations for further robot development and research. In 2007, a nearby start-up company and robot incubator, Willow Garage, sought to build upon these concepts, and initiated a collaborative and open development process of a new software framework. This framework eventually became ROS[ROSc][QGS15]. The framework can be used under the BSD open-source license. Today, ROS comes

in many forms and comprise hundreds of advanced packages, algorithms and drivers, making it applicable for hobbyists, industrial automation, research and everything in between.

3.2.2 Important ROS Concepts

The following descriptions are included in order to provide a complete, self-contained description of the project implementation. Similar descriptions can be found on the official ROS website[1], as well as in any book on ROS (for example [QGS15] or the more comprehensive [Kou16]).

The ROS Graph

A ROS system comprise a set of small programs that communicate with each other through messages. These programs become nodes in the ROS graph. The nodes communicate with each other by publishing and subscribing to topics that form the edges of the graph. A topic must have the format of one of the specific data types provided by ROS. For example, a node which receives temperature data from a thermometer, may publish the data as a topic on the ROS system with the type `sensor_msgs/Temperature`. There are many other data formats, e.g. velocity messages, `geometry_msgs/Twist`; images, `sensor_msgs/Image`; odometry messages, `nav_msgs/Odometry` and so on. Each node in the graph are typically POSIX processes, and the edges are TCP connections[QGS15]. A minimal example of a graph is shown in figure 3.1.

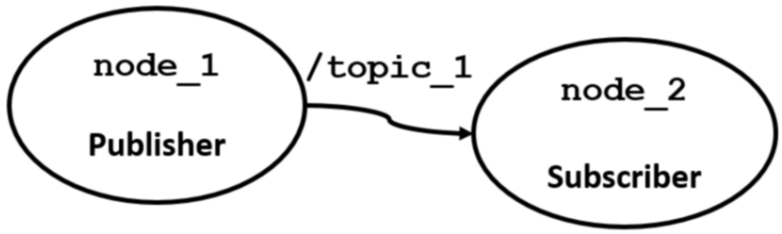


Figure 3.1: A minimal ROS graph. There are two nodes, `node_1` and `node_2`. `node_1` publishes data, i.e. a topic, by the name `topic_1`. `node_2` can receive the data by subscribing to `topic_1`.

roscore

`roscore` is an essensial part of any ROS system as it enables nodes to communicate with each other. An instance of `roscore` must be started before launching any nodes. When a node is started, it will inform `roscore` of which topics it publishes and which

[1] http://www.ros.org/

topics it wish to subscribe to. Then, `roscore` will provide the information which allows the node to form a peer-to-peer connection to other nodes.

tf

`tf`[Foo13] is a coordinate system transformation library used in ROS. Parts of a ROS system can listen to broadcasted transforms in the form of messages, `tf/tfMessage`, which describe a coordinate system transform between one or more parent-child link pairs. `tf` also provides timing information it the messages. This is a very important feature because a node may depend on synchronized sensor streams with many different coordinate frames. Comparing a laser scan with a point cloud that was received seconds ago may lead to errors.

Project Structure and the *catkin* Build System

A ROS project will usually utilize the `catkin` build system. `catkin` replaces *Rosbuild* which is used for ROS Fuerte and earlier (this project uses ROS Indigo). The source code in a ROS system is organized into packages. Each package provides a specific functionality to the system. Some packages can be downloaded and installed from a remote repository, while other packages will be created by the in-house developers for their specific robotic system. In this project, locally created ROS-packages were placed into a *catkin workspace*. This workspace contains the original source code and build specifications. Implementation specific details are provided in chapter 4. As described in [ROSb][2] and [ROSa], a general workspace structure is as follows:

```
workspace_folder/        -- CATKIN WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n

  devel/                 -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    ...
  build/                 -- BUILD SPACE
    ...
```

[2] http://wiki.ros.org/ROS/Tutorials/CreatingPackage

Packages are build by running `catkin_make` from the command line. Each package comes with two files: `CMakeLists.txt` and `package.xml`. In `CMakeLists.txt`, a developer can link to additional libraries, e.g. OpenCV. The `package.xml` file, also known as the package manifest, contains a description of the package and its dependencies. The developer can specify details such as version number, licenses and contact information to the responsible maintainer.

`roslaunch`

`roslaunch`[ROSi] is a ROS package tool used to launch multiple nodes from a single command line. This is useful for larger projects with many nodes, interactions and parameters. Exactly which nodes to launch is defined in XML-files with the *.launch* extension. In a launch file, the developer can group nodes together, pass arguments to the nodes and launch other launch files. Launch files can be launched from the command line as follows (do not include the brackets < ... >):

```
$ roslaunch <package name> <launch file name>.launch <argument1>:=true
```

3.2.3 An Overview of ROS-Related Tools

Robot Modelling In URDF

URDF is an XML-like format for describing robots. The robot description is made up of links and joints. Each link description contains information of, e.g., its shape, inertial tensor, collision boundaries and other attributes. The links are connected to each other by joints.

Visialization in `rviz`

`rviz` is an invaluable tool for visualizing on-line robot behavior. Simply put, `rviz` is created to visualize what the robot sees, and how it plans ahead. Many of the images in the following chapters were obtained in `rviz`.

Simulation in Gazebo

Gazebo[3] is a rigid body real-time simulator with good interfaces to ROS. A developer can build a robot model by using URDF, and spawn this model into a virtual 3D world in Gazebo. The simulator is integrated into ROS by using the `ros_gazebo` package. This simulator was used extensively over the course of this project, and allowed testing of all the implemented features, including SLAM and navigation.

[3] http://gazebosim.org/

3.2.4 Notable Robots Running ROS

PR2 - Personal Robot 2 PR2, developed by Willow Garage is one of the first robots designed to run ROS [QGS15], and also one of the most advanced and capable robots with ROS today. PR2 is build for research and development of service robot applications. The navigation stack used in this thesis has been tested on the PR2. [MEBF+10] describes how the PR2 used the navigation stack to autonomously navigate 42 km (26.2 miles). PR2 is available for sale at the price of $280,000.00[4](2016).

TurtleBot TurtleBot is a cheaper ROS-ready alternative to PR2. It is consists of a mobile base with differential drive, and a shelf system for mounting laptop computers and sensors.

Robonaut 2 Robonaut 2[5], a dexterous humanoid robot, currently resides within the International Space Station (ISS) roughly 400 km above the earth's surface. In 2014, a SpaceX Dracon capsule brought ROS as well as a pair of legs for Robonaut up to the ISS[ROSh]. Robonaut is designed for research on how robots can support the crew in maintaining and operating the space station. A potential application of Robonaut is to perform extra vehicular activities and other maintenance tasks, thus freeing up valuable time for the crew.

Industrial Hardware The ROS-industrial program[ROSd] provides hardware interfaces to various industrial equipment. An example is ABB's IRB-2400, where ROS-industrial provides package for motion planning software (MoveIt!) and trajectory downloading[ROSe].

3.3 Introduction to Sensors in Autonomous Robots

3.3.1 Depth Cameras

Different Methods for Depth Perception

A depth camera can be described as a regular color video camera with the ability to create spatial images. In the context of this thesis, a depth camera can more precisely be described as a RGB-D camera, where the letters RGB-D are short for red, green, blue and depth. In a regular RGB camera, a spatial scene will be projected onto a rectangular pixel grid where each pixel contains intensity values for red, green and blue colors. These pixel values represents the detected scene. A major problem with RGB cameras is the significant loss of information. The information loss is mostly a

[4]https://www.willowgarage.com/pages/pr2/order
[5]ROS in space from ROSCon 2014: https://vimeo.com/106993914

consequence of 3D to 2D projection and digital quantization. RGB-D cameras have the means to reduce this information loss by mapping the pixel values to spatial coordinates. The atomic parts in 3d images are usually represented as points in a point cloud or cubic volumes, also known as voxels.

Different variations of depth cameras will usually fall into one of two categories: active or passive. Passive sensors perceive the surroundings as it is, without actively interfering with the environment as a part of the sensing process. A typical passive RGB-D sensor is the stereo camera. Stereo cameras use a stream of synchronized image pairs to perceive depth. The image pairs are displaced along the horizontal axis, and the depth information is extracted by searching for mutual information in the image pairs. How far the information is displaced from the left to the right image is directly related to how far away from the camera the information source is located.

Active sensors depend on some form of projection onto the surroundings. For depth cameras, the projection is usually in the form of laser or infra red light. In RGB-D cameras it is essential that the projected light is distinguishable from the visible spectrum. The Kinect sensor used in this project is an example of an active RGB-D sensor.

3.3.2 Kinect for Xbox 360

Kinect for Xbox 360 is the RGB-D sensor used in this project. The device was initially intended as a Natural user interface (NUI) for gaming and office applications, and was the first consumer grade sensor to utilize structured light. Possible use cases were inspired by early NUI research at Massachusetts Institute of Technology (MIT) and, later on, the science fiction movie Minority Report, where Tom Cruice interacts with a computer by using hand gestures [WA12]. The Kinect sensor is equipped with a depth sensor, a regular color camera, a microphone array and a tilt motor(figure 3.2). The color camera in combination with the depth sensor forms what is usually referred to as a RGB-D sensor, i.e. a combined color and depth camera (figure 3.2). The exact details on how the sensor works are not publicly available(to the knowledge of this author), but it does involve a variant of structured light. A projector on the Kinect projects an infra red speckle pattern onto the surroundings. This speckle pattern is perceived by the leftmost camera, which is equipped with a visible light filter. The sensor is able to calculate a depth map based on how the projected infra red pattern is distorted by the surroundings.

The Kinect's depth preception capabilities, combined with its relatively low cost and high accessibility has contributed to make the Kinect very popular in research projects related to SLAM and robotics. In the three first years since it's release in 2010, over 3000 papers in well-known journals and proceedings were devoted to research on the Kinect sensor. Roughly 500 of these papers focused on SLAM

or 3d reconstruction[BMNK13]. Some of the other papers focused on some of the weaknesses with the sensor, such as detection of glass surfaces and having several sensors in the same area.

Today, the the Kinect for Xbox 360 has been succeeded by the Kinect for Xbox One, and is now considered to be a legacy device. Those considering to use the legacy Kinect should be aware of that it is becoming increasingly difficult, if not already impossible, to get hold of a new Kinect for Xbox 360.

Natural User Interfaces - Origin of the Kinect

The idea behind a NUI is to make the HMI as seamless and natural as possible. A NUI allows the user to communicate without tools such as a keyboard or a mouse. For decades, NUIs have only existed as ideas, science fiction or research projects. This has changed dramatically over the last ten years, and NUIs can now be considered to be ubiquitous. Today, the most common form of NUIs is the touch screen found in smart phones and tablets.

The Microsoft Kinect sensor was initially designed as a NUI for the Xbox 360 gaming console. The sensor allows users to use gestures and sounds to play console games. Later on, Microsoft has released SDKs, enabling developers to create NUI applications for for Windows.

Kinect Hardware Specifications

Sensor documentation provided online by Microsoft is incomplete and untidy. The reason for this may be that the first Kinect is considered to be obsolete. The following specifications are based on [WA12] and the "Kinect for Windows Programming Guide"[kinc]. There is in fact a distinction between the Kinect for Windows and Kinect for XBOX 360. Only Kinect for Windows can be used in commercial applications[WA12]. Another important issue is that there are compatibility issues between ROS and Kinect for Windows. It is unknown to this author if the compatibility issues have been fixed, but possible hacks have been suggested by the community[kina][kinb].

Sensor specifications are given in table 3.1. Note that the range values may differ from what is available in Microsoft's own SDK, as the shortest distance was measured manually in ROS. In [kinc], the depth values for Kinect for Xbox 360 range from $0.8m$ to $4m$. Other distances are either unknown, too close or too far away. Kinect for Windows can operate in "near mode", i.e. it can measure distances from $0.4m$ to $3m$.

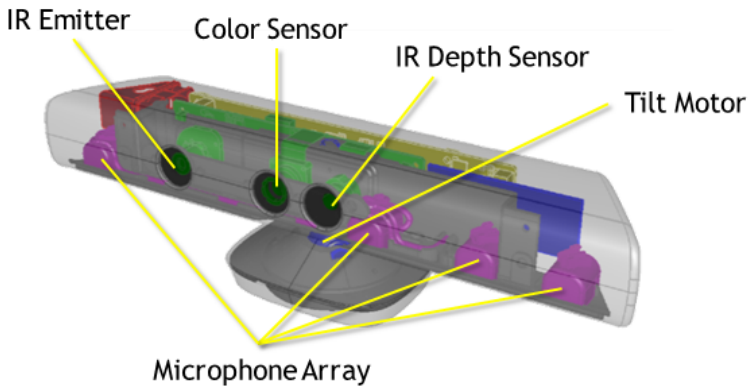


Figure 3.2: Kinect sensor components. (Image credits: Microsoft[kinc])

Kinect for Xbox 360 Specifications	
Viewing Angle	43° vertical by 57° horizontal field of view.
Image Resolution	640x480
FPS	30 Hz (given 640x480 depth and color video)
Minimum Depth (measured)	$\approx 0.5m$.
Maximum Depth	$8m$.
Normal (Reliable) Depth Range	$0.8m < x < 4m$

Table 3.1: Kinect for Xbox 360 Specifications[WA12][kind].

3.4 Simultaneous Localization and Mapping (SLAM)

3.4.1 Introduction to SLAM

SLAM, also known as Concurrent Localization and Mapping (CLM), is a class of solutions to the problem of determining an agents location and pose in an unknown environment, while simultaneously mapping the same environment.

3.4.2 Hector SLAM

Hector SLAM [KMvSK11] is a 2D SLAM approach capable of 3D motion estimation. In its original form, the method is suitable for systems with low-end computational power and size, and for mapping of small environments. 2D SLAM is based on LIDAR scans aligned with the horizontal plane. The full Hector SLAM implementation consist of of a 2D SLAM subsystem loosely coupled and synchronized with an Extended Kalman Filter (EKF) used as a 6DOF pose estimator.

The 2D pose of the LIDAR is estimated through a scan matcher, i.e. the process of aligning the current LIDAR scan with the map generated over the past time steps. The scan matcher was used to provide odometry to RTAB-Map, which is the chosen SLAM system for this robot. Hector SLAM can be downloaded and used in ROS in the form of a prebuild package, `hector_slam`.

3.4.3 RTAB-Map

RTAB-Map is developed by IntRoLab at Université de Sherbrooke in Canada. It is a SLAM system developed for long term operations in large environments. The system is also intended to handle the ”kidnapped robot-problem”, i.e. multi-session mapping. This is useful whenever the robot is shut down and moved to an unmapped part of the same area, where it will start a new mapping session. RTAB-Map is the core feature, besides navigation, that has been integrated into the robot described in this thesis. Some factors which motivated the use of RTAB-Map are:

- It is a SLAM method which requires an RGB-D sensor, for example a Kinect. The problem description for this project requires a vision based solution.
- RTAB-Map has a ROS wrapper, `rtabmap_ros`, which eases the process of integrating it with the mobile robot.
- It includes 3D obstacle detection.
- It has a memory management system intended for large scale multi-session mapping.
- RTAB-Map can be used for object detection. This can be done by linking RTAB-Map to OpenCV and the non-free feature detectors Scale-invariant feature transform (SIFT) and Speeded Up Robust Features (SURF).

The source code and ROS wrapper is currently maintained, and new features and bug-fixes are added regularly. RTAB-Map has two distinctive solutions to the SLAM problem: Visual loop closure detection and a memory management system for large data sets. The following paragraphs provides an overview of how RTAB-Map works. Detailed descriptions of the loop closure detection and memory management approach is provided in [LM13], while the SLAM method is presented in [LM14]. Further details can be found on the project's Github page[6].

[6] http://introlab.github.io/rtabmap/

Graph Based Mapping

RTAB-Map uses a graph structure with nodes and edges to represent the map. New locations L_t, represented by nodes, are continuously added to the system's working memory as time passes. In this method, the graph edges are referred to as *links*. There are two types of links: neighbour links and loop closure links. Each node is a location in the map, and the links contain geometrical transformations between the node locations. Figure 3.3 illustrated the graph concept.

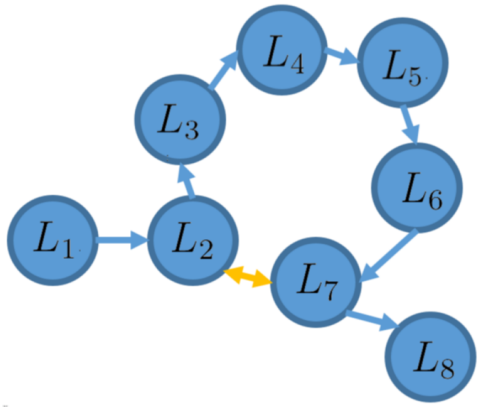


Figure 3.3: Conceptual illustration of a graph created by RTAB-Map over time $1 \leq t \leq 8$. A loop closure hypothesis was accepted at $t = 7$, as shown by the yellow arrow. Feature descriptors in L_2 and L_7 are sufficiently similar to accept this as a loop closure.

Loop Closure Detection

At regular predefined intervals, RTAB-Map will take a snapshot image I and run this image through a feature detector (e.g. SURF or SIFT). The obtained image descriptors will be quantized to a descriptor vocabulary, i.e. a bag-of-words. The bag of words associated with an image, constitutes the signature of location L, i.e. the location at which the image was taken. When the new location, L, is added to the graph, RTAB-Map will update an a posteriori probability density function for each loop closure hypothesis S by means of a Bayes filter. A loop closure hypothesis may be accepted if the belief that the robot is at a new location is below a threshold T_{loop}. Then a link in the traversal graph will be created to represent the closed loop, as illustrated in figure **??**.[LM13].

On-line Mapping of Large Environments

As the graph of locations grow, the time it takes to check for loop closures will also grow. In RTAB-Map, this problem is handled by a memory management scheme,

where memory is split into Short Term Memory (STM), Working Memory (WM) and Long Term Memory (LTM). Loop closure detection is performed on locations in WM. To satisfy the real time constraint, set by the rate of adding new locations to the graph, RTAB-Map will tranfer images from WM to LTM. An accepted loop closure hypothesis will trigger a transfer from LTM bach to WM.[LM13].

3.4.4 RGBD SLAM and Octomap

Octomap[HWB+13] is another 3D mapping framework available for ROS. Similar to RTAB-Map, Octomap can also be used as a standalone version.

Maps are represented by memory efficient Octrees where each leaf node represents a cube, or voxel, in the volumetric map. The voxel can be either occupied, free or unexplored. The volume of the cube is determined by how deep in the tree the leaf node is located. In a ROS graph, the Octomapping is performed by the node `octomap_server`. This node will subscribe to point cloud messages `sensor_msgs/PointCloud2`, and return volumetric occupancy maps, i.e. Octomaps.

There are several approaches to SLAM which uses Octomaps. An example that stands out in the context of ROS is[EHS+14]; a SLAM approach which depends on a RGB-D sensor, and relies on Octomap for efficient map storage.

This mapping framework was not used in this project in order to limit the project scope, and because the alternative RTAB-Map was associated with less uncertainty.

3.5 Autonomous Navigation

ROS provides a pre-built navigation stack for 2D navigation. The navigation stack can plan a path and send velocity commands to the mobile base controller based on sensor input, a goal pose, a map and the frame of the `base_link`. Figure 3.4 shows how the internal components of the node `move_base`, and how it interacts with the rest of the ROS system.

The navigation stack publishes velocity commands for translation and rotation in the $xy-plane$. Velocity commands are published in the form of a `geometry_msgs/Twist` message with the default topic name "`cmd_vel`". Mobile bases must be constructed as either holonomic or differential drives.

The two following subsections introduce the global and the local costmaps respectively. These costmaps are in fact grid maps where each square on the grid can be viewed as a node in a graph. This graph can be used by path finding algorithms to search for a shortest path. These maps are both based on the ROS package `costmap_2d`. This package will inflate an area in a radius around the detected obstacles, in which

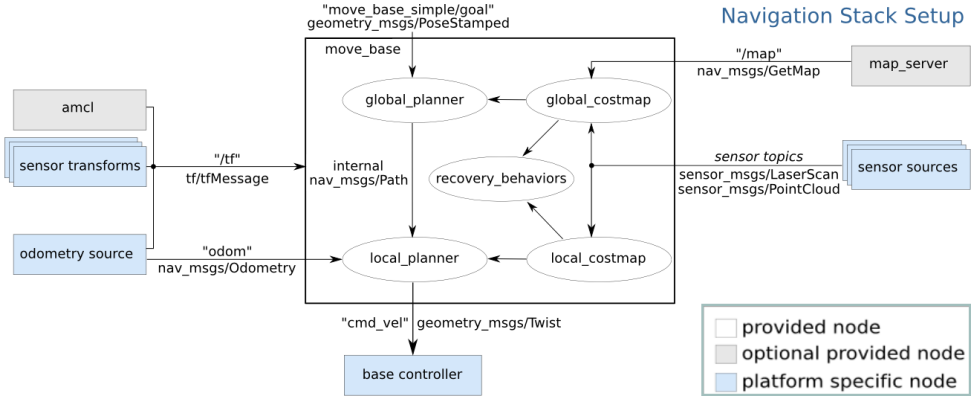


Figure 3.4: `move_base` and the navigation stack. (Image credits: ros.org)

traversal is associated with a certain cost. By inflating the obstacles, which in some sense is similar to increasing their size, the path planners can now treat the entire robot as a single point. The point represents the origin of the robots base frame.

3.5.1 Global Planner

The ROS navigation stack is equipped with a basic *global planner*. This global planner is fitted with a set of basic path planning algorithms: e.g. Dijkstra and A*, where Dijkstra is the default option. Global path planning is performed on the global costmap which in turn is based on the 2D grid map published by a map server. A pre-mapped area is not strictly necessary for the global planner to plan a path. Instead it will just plan a naive path which can be corrected by the local planner as the robot moves along the planned trajectory.

3.5.2 Local Planner

Actual velocity commands from `move_base` are published by the *local planner*. The local planner will receive the global plan from the global planner, and calculate a new trajectory based on currently observed obstructions as well as the robot footprint and kinematics. Local obstructions are expressed in a dynamic grid map, i.e. the local costmap, which is based a subset of the global map combined with real time sensor data. The dynamic local cost map enables the robot to avoid temporary and moving obstacles.

The local planner can use either the *Trajectory Rollout* or *Dynamic Window Approach (DWA)* algorithm for trajectory planning. Further elaboration on these methods is outside the scope of this thesis.

3.5.3 Recovery Behaviors

When the robot becomes stuck, it can be configured to perform a set of recovery behaviors. The default recovery behaviors available to `move_base` are to clear the costmap, i.e. to remove obstacle information from the costmap, and to rotate in place, in the hope of discovering a new clear path. A typical sequence of behaviors is to clear the costmap, try to locate a path and then attempt an in place rotation before doing a second attempt to plan a path. If the entire series of recovery actions fails to reveal a clear path, the `move_base` node will abort and consider the goal state to be infeasible[Kou16].

Chapter 4
Implementation

4.1 Introduction

This chapter presents four implementations that were developed during this project:

Robot software in ROS

This is the core implementation that connects all the other pieces together. The robot's operating system runs on a laptop computer which is placed on the robot. This system is responsible for the core functionality of the robot, which includes sensor and actuator management, mapping, navigation and manual control.

Motor Control Firmware

Motor control firmware, running on an XMEGA A3BU board, provides an interface between the ROS computer and each wheel motor. Velocity commands from ROS are translated into wheel speed commands.

Android Application

A supporting tool intended to function as a remote control for the robot. The implementation presented here enables the user to control the robot from an Android device via a Bluetooth connection.

Operator Control Station

A simple Operator Control Station (OCS) based on Qt enables an operator to control the robot via a wireless TCP/IP connection. The OCS can display a live video stream from the Kinect sensor. The purpose of this implementation is to learn how ROS can be connected to the outside world.

4.2 Hardware Setup

The implementations listed in the introduction created a need for additional hardware. Extensive modifications had to be made to accommodate these additions. Some improvements with respect to safety were made as well.

4.2.1 Second On-Board Computer and New Rear Compartment

The robot was already fitted with an on-board computer running Windows 7. Using this on-board computer was not an option, due to the following reasons:

- It lacked the computational power required for this implementation. Both the Gazebo simulator and RTAB-Map are computationally intensive.
- The hard-drive was full. No disk space for the Linux partition required by ROS was available.
- Two parallel master's projects were using the robot hardware, and both project implementations required an on-board computer. Sharing the hardware would have been very time consuming.

For these reasons, this author decided to develop the robot software on a second on-board computer.

The previous robot chassis did not facilitate any good wiring solutions. The cables belonging to the various equipment on the robot would often result in a huge tangle of cables and wires. To accommodate the second on-board computer and to facilitate a tidier cable management, it was decided to build a new rear compartment where the equipment could be placed[1].

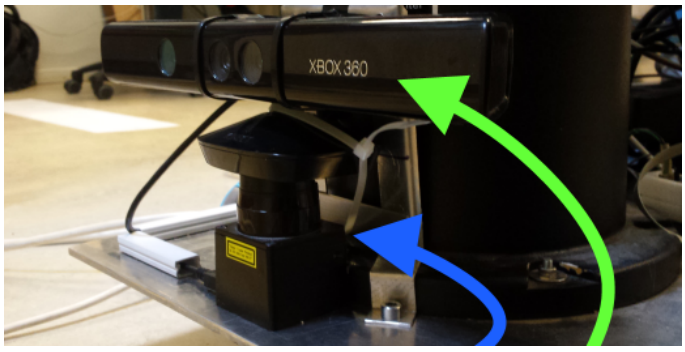


Figure 4.1: Sensor locations for LIDAR and Kinect.

[1] This was a collaborative effort done together with this author's colleague, Ole Magnus Siqveland

4.2.2 Sensor Calibration and Setup

Only support for the Kinect and the LIDAR sensor was integrated into this system. Odometry from the two wheel encoders was not included because of time constraints on the project. Figure 4.2 illustrates how the sensors and the wireless router is connected to the ROS computer and how they are supplied with power.

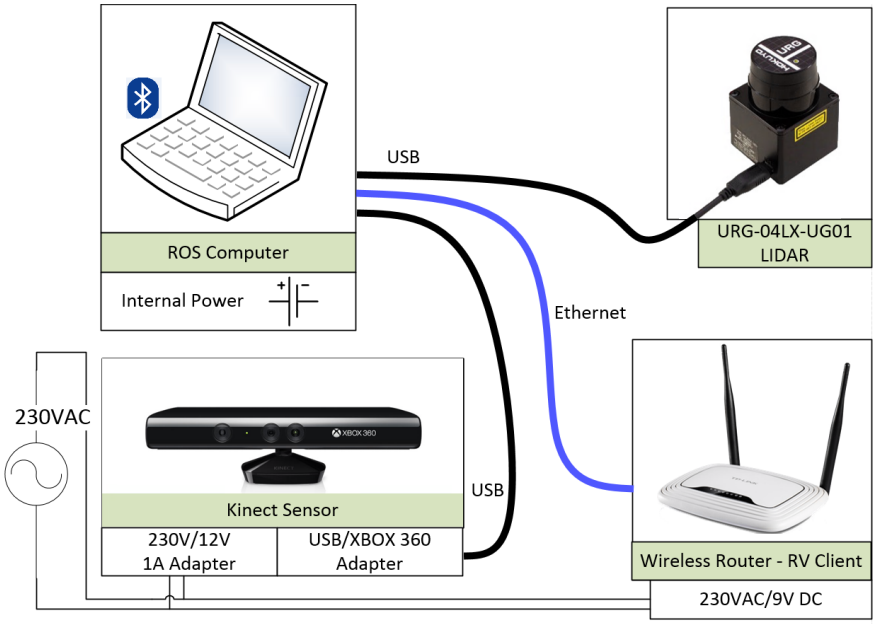


Figure 4.2: Sensor and power supply connections.

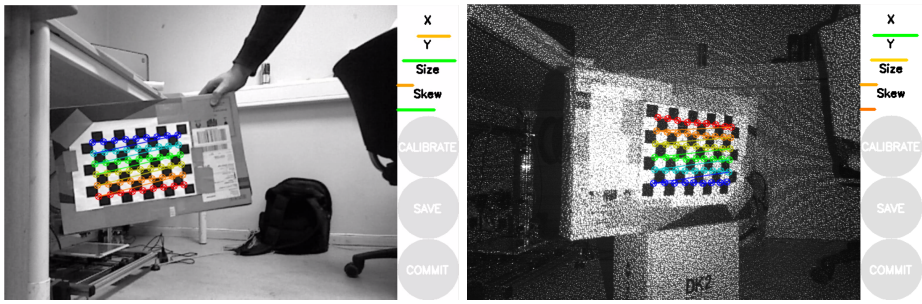
Calibrating both the Kinect and the LIDAR is a straight forward procedure with ROS. The Hokuyo LIDAR will in fact be calibrated automatically when the node is launched. Calibrating the Kinect is actually not strictly necessary because the lens distortion is very low. However, because there already is a calibration tool available in ROS[2] that is easy to use, there is no good reason to not calibrate. Calibration is highly recommended in the RTAB-Map configuration tutorial for ROS[rta]. The calibration procedure is as follows:

1. Print out a chessboard pattern and tape it to a flat surface. It is beneficial to use a large paper size, for example A3, to make the pattern easier to detect over a larger range of distances.
2. Calibrate the RGB camera by using the calibration tool. Use the RGB video stream.

[2]ROS calibration guide: http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration

3. Calibrate the IR camera by using the calibration tool. Stream from the IR camera this time. For this procedure, it is recommended to cover the IR projector, because the IR speckle pattern makes it difficult to detect the chessboard (figure 4.3b).

The calibration program needs to know the number of inner corners of the chessboard pattern, and the size of the squares. The chessboard used in this project has 6 by 9 inner corners and the size of the squares is $0.0275m$. Larger pattern squares will make it easier to detect the chessboard pattern over a larger range of distances.



(a) RGB camera calibration. The camera can be calibrated when a sufficient number of samples have been obtained.

(b) IR camera calibration. The chessboard pattern will be difficult to detect, because the IR projector is not blocked.

Figure 4.3: Depth camera calibration.

4.2.3 Power Supply and Battery Safety

This system requires only a subset of the equipment mounted on the robot. A 24V battery used by M. Berg and P. Aspunvik([Ber13] and [Asp13]) is replaced by a $230VAC/24VDC$ converter. Besides the AC/DC converter, the Kinect and wireless router are the only components which require a supply of $230VAC$. A possible power supply setup is shown in figure 4.4.

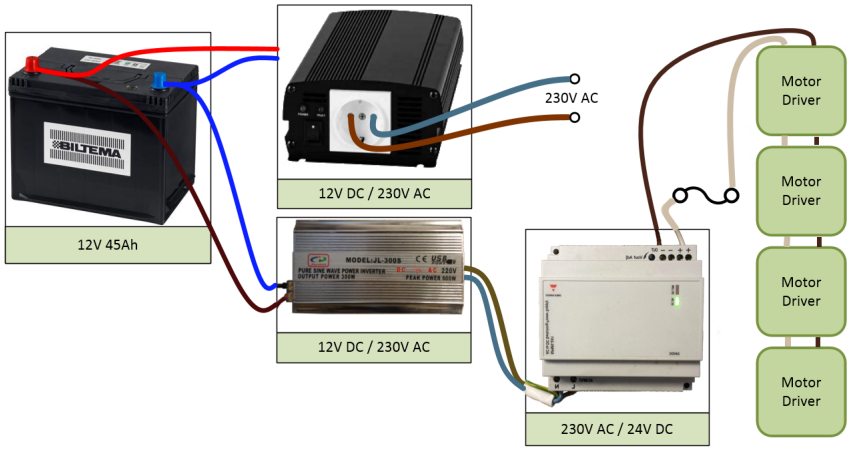


Figure 4.4: Example of a feasible power supply setup.

The car battery is now placed in the lower shelf in the rear compartment where it is surrounded by aluminium. This presents an unacceptable risk, because a short circuit is much more likely in this location if the battery poles are uncovered. To reduce the possibility of having a short circuit, an isolating rubber pad was glued to the inner wall and roof next to the battery in the rear compartment. New isolated battery connections provide additional protection.

4.3 ROS Integration Overview

The process of integrating ROS with the mobile robot platform was influenced by chapter 16 in [QGS15], "Your Own Mobile Robot". The steps suggested by [QGS15], are:

1. Decide on ROS message interface. Section 4.6.
2. Write interfaces for the motor drivers.
3. Create a description of the physical structure and properties of the robot in URDF. Section 4.4.
4. Extend the model to allow simulation in Gazebo.
5. Publish coordinate transform data via *tf* and visualize it in `rviz`.
6. Add sensors, with driver and simulation support.
7. Apply algorithms for navigation and other functionality. Section 4.10.

Packages within `mar_catkin_workspace`	
`mar`	Launch files for the real robot hardware and for RTAB-Map.
`mar_gazebo`	Launch files for the simulated robot and for RTAB-Map.
`mar_description`	URDF files for both the simulated and the real robot.
`mobile_platform`	Programs for handling and processing velocity commands. One such node serves as an interface to the motor control card.
`mar_2dnav`	Configuration and launch files for both the real and simulated robot.
`bluetooth_server`	A node that serves as a Bluetooth server based on the Qt5 API.
`server_wifi`	Contains the node responsible for communicating with the OCS.

Table 4.1: List of custom made packages.

The robot software implementation is placed within a `catkin` workspace (see section 3.2.2) that contains all the project specific files that are necessary to build and run the robot system. The overarching file system is shown in figure 4.5. Note that there are several references to the Three Letter Acronym (TLA) ”mar”, which is short for Mobile Autonomous Robot (MAR).

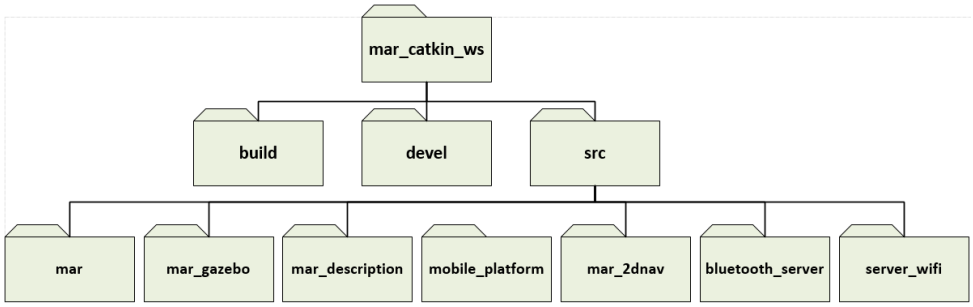


Figure 4.5: Overarching file system. The ROS packages are located within `src`.

Each package will be introduced in the following sections. The purpose of presenting the file systems is to clarify which parts of the system that was implemented by this author. A short description of each package is given in table 4.1, and their place in the workspace hierarchy is shown in figure 4.5.

4.4 Modeling

A robot model will serve two purposes in this implementation. First of all, the system needs a definition of how the sensor inputs are placed with respect to the `base_link`. The origin of `base_link` is associated with a coordinate frame. This frame, and any other frame in ROS, is right handed, i.e. the positive x direction is *forwards*, positive y points *left*, and positive z is *up*. The robot pose will be based on the transformation between the world and the `base_link` frame.

The second purpose of the model is simulation. Being able to simulate the robot system has been invaluable throughout this project. The robot model is represented in an XML-based modeling language called URDF (Unified Robot Description Format). There are two *.urdf* files within the package `mar_description`; one for the simulator and one for the real robot hardware. The following sections presents how these models were defined.

4.4.1 Physical Dimensions

Step one in building the model was to define its geometrical shape. The current model shape consists of several links. Each link is defined as a shape and a size. The links are connected together by joints that define the coordinate transformation between the links. All links were modeled as either cuboids or cylinders, in order to simplify and speed up the modeling process. All joints are static except for the wheels which are continuous joints. For simplicity, the robot arm is modeled by a dummy link with the shape of a cylinder.

After defining the model shape, it is time to add some additional physical attributes to each link. Each link requires an inertia tensor in order to simulate the model. It is also useful to define a collision volume for each link. In this model, the collision volume is equal to the geometric shape of the link without exceptions. Inertia tensors for each shape is based on equations 4.2 or 4.3.

The inertia tensor:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{4.1}$$

Inertia tensor for a solid, uniform cylinder where the radius r is measured in parallel

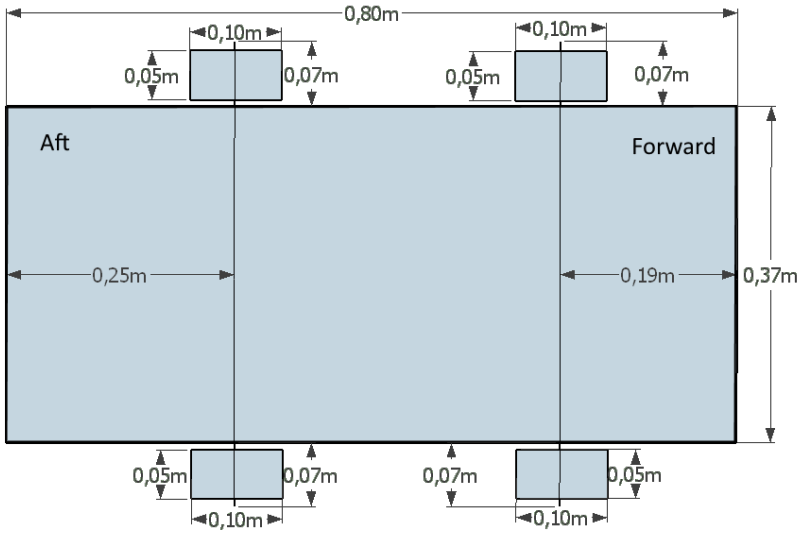


Figure 4.6: The robot footprint. Dimensions are used for the navigation planners and for modeling.

to the $x - y$ plane, and h is parallel to the z axis:

$$I_{cylinder} = \frac{1}{12} m \begin{bmatrix} (3r^2 + h^2) & 0 & 0 \\ 0 & (3r^2 + h^2) & 0 \\ 0 & 0 & 6r^2 \end{bmatrix} \tag{4.2}$$

Inertia tensor for a solid, uniform cuboid. The subscript of l indicates which axis l is measured along:

$$I_{cuboid} = \frac{1}{12} m \begin{bmatrix} (l_y^2 + l_z^2) & 0 & 0 \\ 0 & (l_x^2 + l_z^2) & 0 \\ 0 & 0 & (l_x^2 + l_y^2) \end{bmatrix} \tag{4.3}$$

The mass of each link is guesstimated. Consider the `base_link` as an example. The base link was measured to be 5 mm thick, 37 cm wide and 80 cm long. Assuming that the density of aluminium[3] is $2.7 g/cm^3$, the mass of this link was calculated to be $\approx 4kg$. Mass and volume calculations for the other links are given in chapter B. The base link was defined as follows:

[3] https://en.wikipedia.org/wiki/Aluminium

```
<link name="base_link">
  <visual>
    <geometry>
      <box size="0.8 0.37 0.005"/>
    </geometry>
    <material name="silver"/>
  </visual>

  <collision>
    <geometry>
      <box size="0.8 0.37 0.005"/>
    </geometry>
  </collision>

  <inertial>
    <mass value="4" />
    <origin xyz="0 0 0" />
    <inertia ixx="0.04564" ixy="0.0" ixz="0.0"
      iyy="0.21334" iyz="0.0"
      izz="0.25879" />
  </inertial>
</link>
```

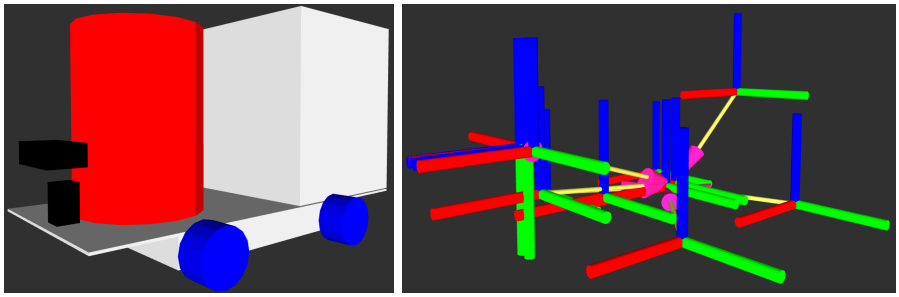
4.4.2 Connecting the Links

Robot links are connected by joints. A joint defines a translation and rotation from the coordinate frame of a parent link to that of a child link. Each joint has two attributes: *name*, for example ”base_link_to_left_wheel” and type, for example ”prismatic” or ”continuous”. All joints in the mobile robot are static, except for the wheel joints which are continuous. Correct joint transformations is very important when placing the sensors (figure 4.8). A discrepancy between the real sensor-to-base transform and the modeled transform will cause misaligned sensor input.

4.5 Simulations

Robot simulation was done in Gazebo; a simulation tool with good interfaces to ROS. The same ROS graph was used for both the simulated and real version of the robot, except from the sensors and actuators, and some minor parameter changes.

To properly simulate the robot, Gazebo needs a way to simulate the differential drive and the sensors in addition to the physical description of the robot. An expanded



(a) Complete URDF model when viewed in `rviz`.

(b) URDF model in `rviz` with all link frames and transformations.

Figure 4.7: URDF model.

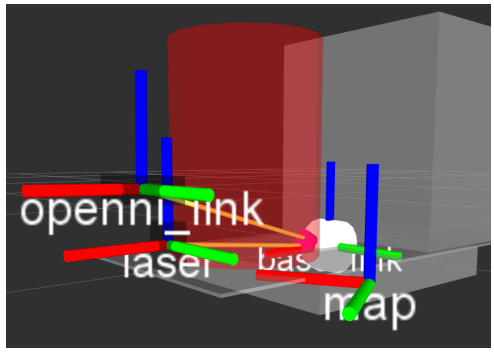


Figure 4.8: Robot model with frames for laser, Kinect, robot base and map.

URDF description of the robot, `mar_model_sim.urdf`, intended for Gazebo was placed within the `mar_description` package (figure 4.5).

4.5.1 Robot Description Plugins

The URDF model in `mar_model_sim.urdf` was expanded with plugins that simulate the motor control card, the LIDAR and the Kinect. The plugins are provided by Gazebo and are intended for use in ROS.

The motor controller is simulated by the `skid_steer_drive_controller` plugin. This plugin was preferred over the `differential_drive_controller` because it supports four wheel joints instead of two, and because the skid steering controller was considered to be good enough.

Attributes for gazebo's sensor plugins are based on the technical specifications[hok] for the real sensors.

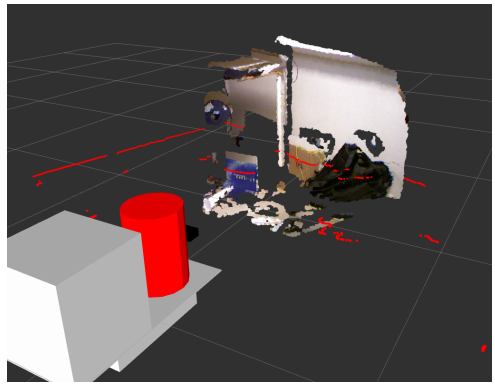


Figure 4.9: Sensor input placed with correct transformations from `base_link`.

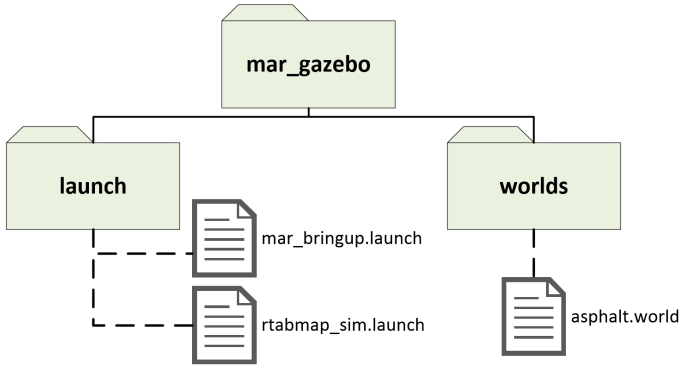


Figure 4.10: Folders and files specific for the simulator.

4.6 ROS Nodes for Motion Control

All nodes for motion control are located within the package `mobile_base`. This package is organized as shown in figure 4.11.

4.6.1 Velocity Command Flow

There are four sources that can generate velocity commands for the robot. Common for all of these is that they use the message format `geometry_msgs/Twist`. The names for each of the four velocity messages are as follows:

/cmd_vel_loc Local keyboard input.

/cmd_vel_wifi Wireless teleoperation from the OCS.

/cmd_vel_bt Wireless teleoperation from a hand-held Bluetooth device.

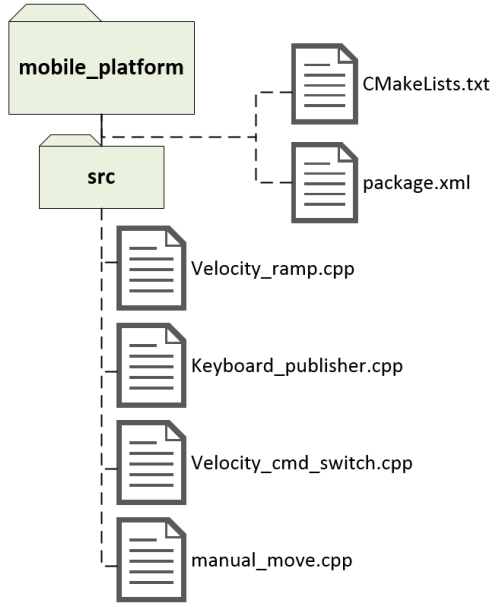


Figure 4.11: Files within the package `mobile_platform`.

/cmd_vel Commands from the navigation stack in ROS. `/cmd_vel` is the default velocity command topic for the navigation stack.

The third topic `/switch_setting` is a command used to select the velocity command to be passed all the way to the motor control card, or Gazebo in the case of a simulated session. The message flow between source and sink is illustrated as a ROS graph in figure 4.12.

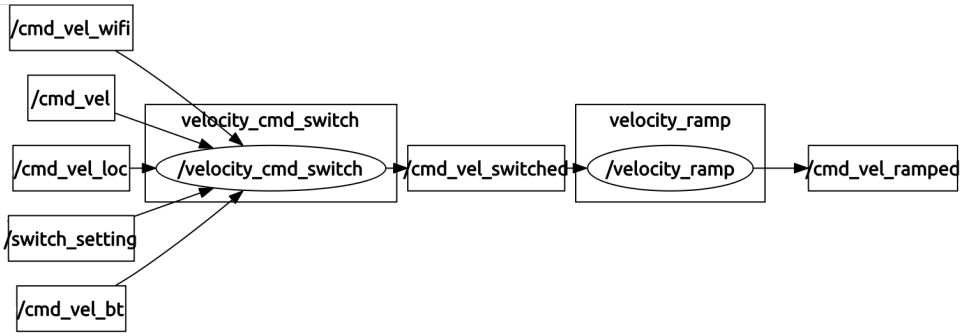


Figure 4.12: Nodes and topics for motion control. (see figure 3.1 for an explanation of this figure).

Keyboard Publisher

The first velocity command publisher was implemented as a local keyboard listener. Keystrokes are registered by `getchar()` in a continuous while loop. The terminal behavior has been altered through a call to `system("/bin/stty raw")`, which will pass each keystroke directly to `getchar()`, without requiring the user to press Enter. This will rented Ctrl +C useless, as the combination will be split into two inputs. A distilled keyboard listener loop is listed below:

```
system ("/bin/stty raw");
while (ros::ok()){
  std_msgs::Char msg;
  keypress = getchar();
  if(keypress == 27) // Press 'Esc' to exit while loop.
    break;
  msg.data = keypress;
  key_input_pub.publish(msg);
}
system ("/bin/stty cooked");
```

After a user has entered Ecs, the program can be exited as normal with Ctrl +C.

Velocity Ramping

Rapid changes in the velocity command may cause excessive wear and tear on the equipment. The node `velocity_ramp` has been added to ensure that the velocity command values will change smoothly. Implementation of this node is heavily based on example code from [QGS15]. The example code was translated from Python to C++, and adapted to fit into this particular system. Figure 4.13 shows an example of how `velocity_ramp` works.

4.6.2 Motor Control Card Firmware on XMEGA A3BU

XMEGA A3BU is an evaluation board developed by Atmel. The The implementation presented here is an adaptation of Petter Aspunviks implementation [Asp13]. The following paragraphs presents the most significant firmware changes that were made.

The firmware will now receive angular and linear velocity commands based on the `geometry_msgs/Twist` message format in ROS, and translate these into the command format used by each motor. Speed settings for each motor is based on Pulse Width Modulation (PWM).

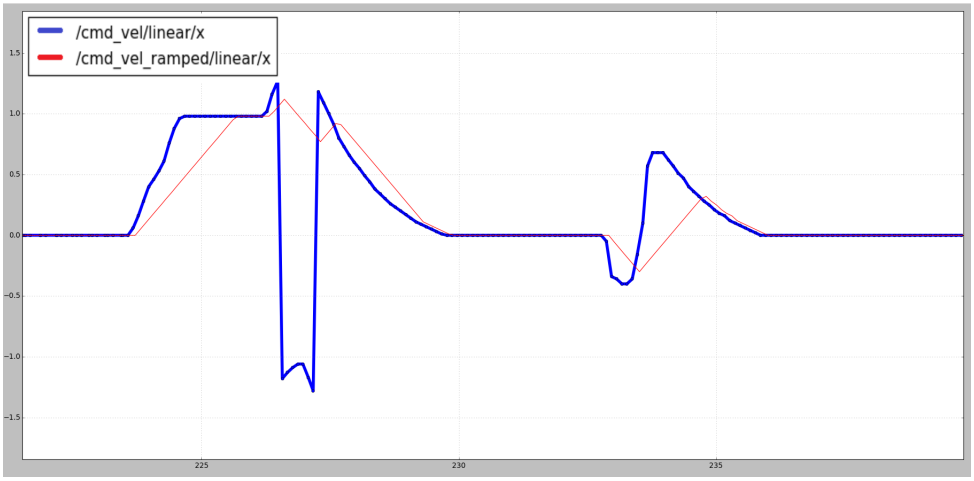


Figure 4.13: The node `velocity_ramp` limits the rate of change of the velocity command sent to the motor control card. The blue line represents commands entering `velocity_ramp`, while the red line shows the acceleration constrained output command.

There were two requirements for this implementation:

1. When velocity commands from the operating system are either absent or incomplete, the robot shall stop.
2. The program shall translate linear and angular velocity commands into wheel commands.

The connections in figure 4.14 are the same as in [Asp13], except for the installation of more secure connections under the robot. The old connections were insecure, and the risk of short circuits was substantial.

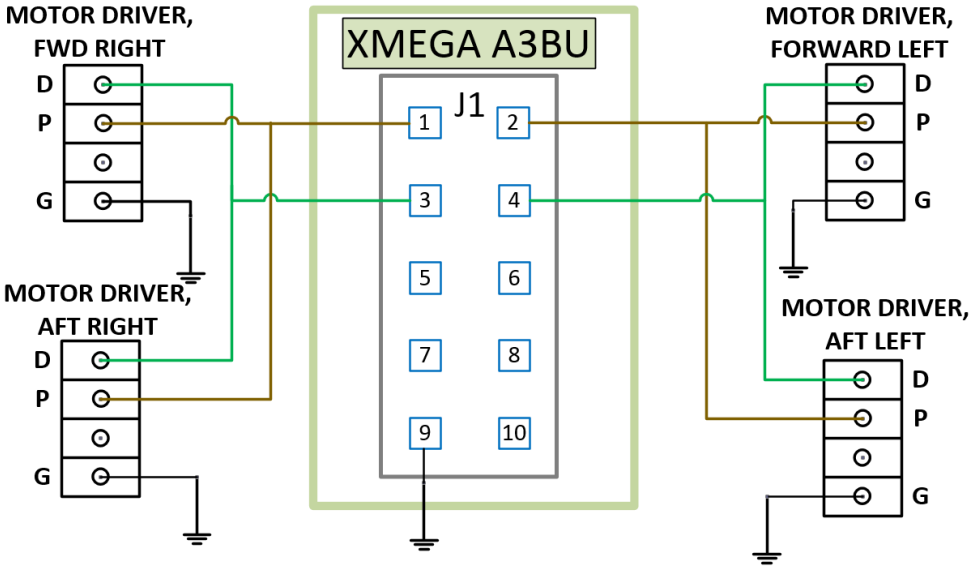


Figure 4.14: Connections between each wheel motor driver and the motor control card, XMEGA A3BU. The connections are unchanged from [Asp13], except for some improved connection for better short circuit prevention.

ROS-Motor Driver Communication

Communications between the motor control card and ROS is done via a USB serial port. The node `motor_driver_interface` within the package `mobile_platform` is responsible for transmitting velocity commands via the comport. The node is hard-coded to communicate via `/dev/ttyACM1`. The velocity update transmission cycle is shown as a sequence diagram in figure 4.15. The transmission cycle is initiated each time `motor_driver_interface` receives a velocity message on the topic `/cmd_vel_ramped`.

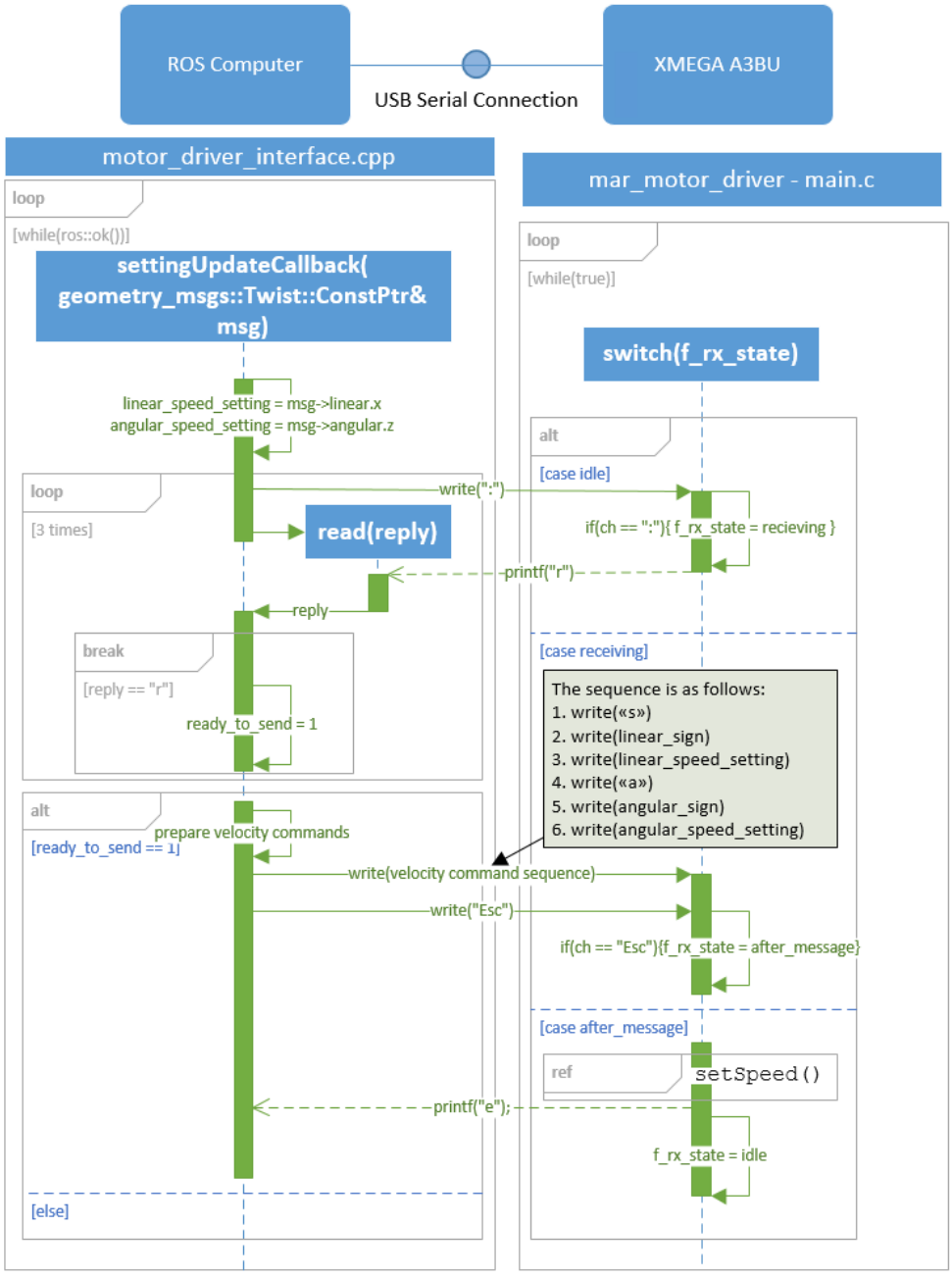


Figure 4.15: Velocity command transmission sequence from the `motor_driver_interface` in the ROS computer to the motor control card (XMEGA A3BU).

From Velocity Commands to Wheel Commands

Within ROS, velocity commands are passed around between nodes in the form of the message `geometry_msgs/Twist`. This message type can be viewed as a *struct* with the following contents:

```
Vector3  linear
Vector3  angular
```

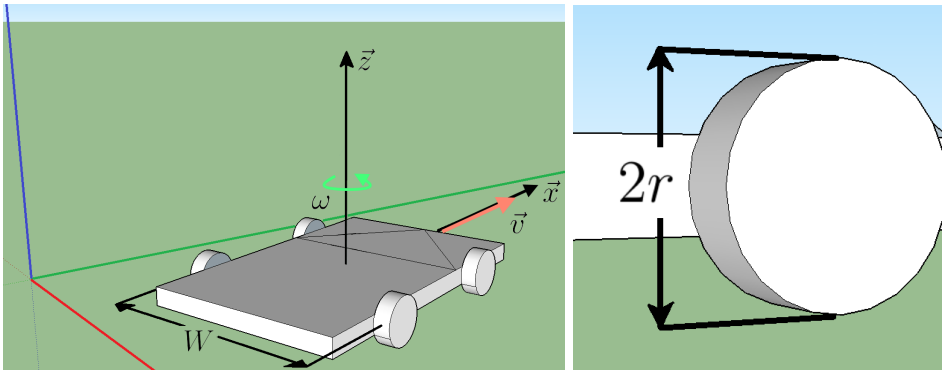
where each vector contains float values for the directions x, y and z with respect to the robot's base frame. Because of the motion constraints of this robot, only `linear.x` and `angular.z` are of relevance, and the data which is passed to the motor control card (XMEGA A3BU) is therefore limited to these two values. The motor control card must now translate the linear and angular velocities into wheel speeds. Next, these speeds must be related to a duty cycle for the PWM signal which controls each of the four motors.

To perform the translation, it is assumed that the mobile base can be described as a vehicle with differential drive steering. Wheel commands will only distinguish between left or right - not front or aft. Equations of motion which relates angular and linear velocity to wheel velocities can be found in[Coo11], and are shown below:

$$\omega = \dot{\psi} \tag{4.4a}$$
$$v_{left} = \omega(R - W/2) \tag{4.4b}$$
$$v_{right} = \omega(R + W/2) \tag{4.4c}$$

W is the spacing between the wheels as shown in figure 4.16a. In [Coo11], the parameter R represents the instantaneous radius of curvature of the robot trajectory. This mouthful will be substituted by the linear velocity v in the following equations, because $v = R\omega$ (similar to the linear speed of a wheel). This yields two equations for the wheel speeds, v_{left} and v_{right}, based on angular and linear velocity, w and v.

$$v = R\omega \tag{4.5a}$$
$$v_{left} = \frac{2v - \omega W}{2} \tag{4.5b}$$
$$v_{right} = \frac{2v + \omega W}{2} \tag{4.5c}$$



(a) Differential drive parameters. (b) Wheel diameter.

Figure 4.16: Parameters for differential drive kinematics. Note that the frame vectors $\vec{z}$ and $\vec{x}$ refer to the base frame of the robot in this case, and not the world frame.

A wheel command update cycle will begin, on the motor control card, when a new velocity setting is received from ROS. After receiving the "end-of-message" token, "`Esc`", the receiver state machine will switch to the `after_message` state (see figure 4.15). Three things will happen within the `after_message` block of the state machine:

1. `setSpeed(f_linear_speed_setting, f_angular_speed_setting);`
2. Switch state variable is set to `idle`, making the receiver ready for a new cycle.
3. `tc_restart(&TCC1);`

`tc_restart(&TCC1)` will reset the timer for `ovf_interrupt_callback`. This interrupt callback is intended to set the wheel speeds to zero if a steady stream of velocity commands are absent for whatever reason. This is done by entering a loop that will decrement the speed settings to zero, in order to ensure a smooth stop. The speed update cycle is listed below. `calculate_left_wheel_speed` returns a wheel speed as calculated in equation 4.5b, before the new motor speed is set.

```
void setSpeed(int16_t linear_speed_setting,
              int16_t angular_speed_setting)
{
    int16_t left_speed_setting =
           calculate_left_wheel_speed(linear_speed_setting,
                                      angular_speed_setting);
    int16_t right_speed_setting =
```

```
            calculate_right_wheel_speed(linear_speed_setting,
                                        angular_speed_setting);
    left_set_wheel_speed(left_speed_setting);
    right_set_wheel_speed(right_speed_setting);
}
```

4.7 Operator Control Station (OCS)

A HMI is an integrated part of any remotely operated maintenance system. The OCS allows an operator to control and monitor the robot through a graphical user interface. To communicate with the OCS, the robot will set up two servers: a `web_video_server` and a node called `server_wifi`, where `web_video_server` is a finished ROS package available online.

`server_wifi` is a custom made ROS node which receives velocity commands over a wireless TCP connection from the OCS. The TCP socked implementation is based on sample code from a tutorial on socket programming [tcp], which has been modified to fit into this specific project. The TCP server is structured much in the same way as the motor control firmware, with a receiver state machine. After a new velocity command is received, `server_wifi` will publish a new `geometry_msgs/Twist` message with the topic name `/cmd_vel_wifi`.

`web_video_server` is a ROS package created by the Robot Web Tools community. In this project, `web_video_server` is used to publish a colour image from the Kinect to a URL address. The video at this address will in turn be received by the OCS. The chosen image stream is initially published by `openni` under the topic name `/openni/rgb/image_raw`. Because of bandwidth constraints, the video frame rate is throttled down from $30Hz$ to $10Hz$. `web_video_server` will then place the video at the specified URL with a reduced quality and size.

The OCS itself will connect to `server_wifi` by means of a TCP socket, and transmit velocity commands based on the position of the `controlNode` object, shown as a yellow ball at the center of figure 4.17.

4.7.1 Graphical User Interface

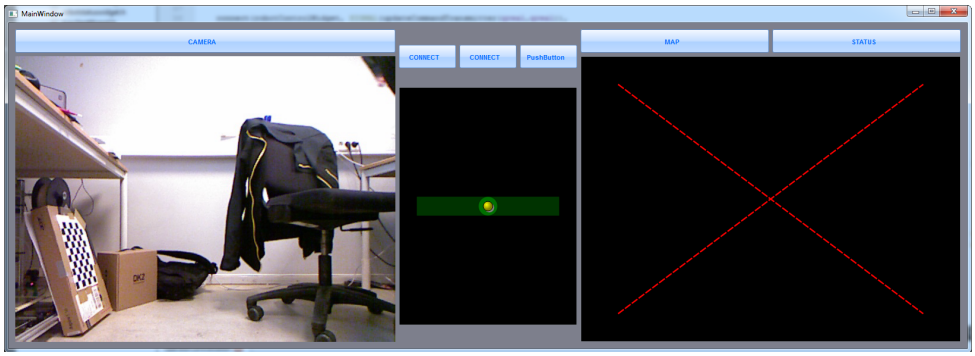
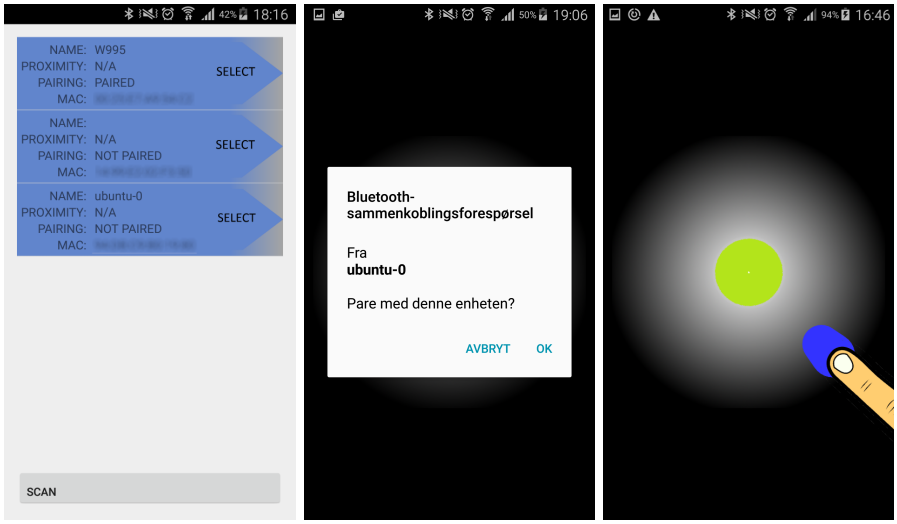


Figure 4.17: Operator Control Station (OCS) HMI. The current skeleton implementation displays live video from the robot. The operator can steer the robot by moving the yellow ball in the center screen.

The OCS is created with Qt5, a cross-platform framework with a vast set of APIs and a large framework for Graphical User Interface (GUI) design and development. The GUI was planned to consist of three screens: two multi function displays on each side of a control screen. In its current form, only the camera display on the left side is implemented, and the user can interact with a yellow ball on the center screen to control the robot. Code implementation for the center display, `RobotControlWidget` is based on sample code from the "elasticnodes" example of the Qt Toolkit[4]. The yellow ball is defined by the `ControlNode` class. The position of the node is subjected to mass-spring-dampener dynamics, to ensure that the ball will return to center when released. Two force vectors F_x and F_y are calculated and applied to the motion equations of the `controlNode`. Force calculations are performed at a constant rate which defines Δt used in the position update calculation.

[4]http://doc.qt.io/qt-5/qtwidgets-graphicsview-elasticnodes-example.html



(a) First activity with device list.

(b) The user is prompted to pair with the robot.

(c) Controlling the robot with the stick.

Figure 4.18: A typical use case for "Robot Leash".

4.8 The Hand Held Remote Control - *Robot Leash*

Because the OCS is only partially implemented, an operator will not have access to all the features on the robot. In addition, as a safety precaution a person should be close to the robot at all times, and be ready to pull the plug. Furthermore, it is hard to control a moving robot through the on-board keyboard. These problems were countered by the Android-based remote control, *Robot Leash*. This application allows an operator to steer the robot from a mobile device via a Bluetooth connection. Details of the implementation are not included. This section will rather focus on how the mobile application interacts with the robot. A typical use case is as follows (figure 4.18):

1. The robot is online, and a discoverable Bluetooth server is running on the robot computer.

2. An operator wishes to control the robot, and starts "Robot Leash" on his/her Android device. The operator can now scan for devices within range.

3. After clicking the *scan* button, the robot is discovered. When the operator selects the discovered device, he/she will be prompted to pair with the robot.

4. The smartphone and the robot is now paired, and velocity commands from the blue control stick are passed to the robot via Bluetooth. The velocity command switch on the robot, `velocity_cmd_switch`, must be set to `/cmd_vel_bt`.

4.8.1 Application Structure

The GUI used to control the robot (figure 4.18c) is implemented in OpenGL ES2. To save time, the code structure behind the graphics is an adaptation from a coding sample in the book *OpenGL ES 2 for Android*[Bro13]. The same approach is applied to the portion of the app which manages the Bluetooth connection, i.e. the file `BluetoothConnectionService.java`. The code within this file is taken from a Bluetooth chat sample[5], provided by Google.

4.8.2 Interaction With the Robot

Communication between robot and smartphone is done over a Bluetooth connection with a server-client pair. The robot will set up a Bluetooth server, exposing a service with a Universally Unique Identifier (UUID). The Android application can now connect to this specific service. To speed up the implementation process, the service UUID has been hard coded into the Android application. It is recommended to implement a more elegant solution later. Figure 4.19 shows how a touch gesture by the user is transferred to the robot as velocity commands.

Android and Qt offers the option to choose either a secure or non-secure connection. The Android reference states that an insecure connection ”not have an authenticated link key”[and], making it vulnerable to man-in-the-middle attacks. However, for reasons of ease of use, the connections used here are set to non-secure. No further functionality was added besides the ability to send transmit velocity commands. Additional changes were only minor, such as disabling screen rotations when controlling the robot.

http://developer.samsung.com/technical-doc/view.do?v=T000000117

4.9 Mapping - Setting Up RTAB-Map

This robot is using Real-Time Appearence Based Mapping (RTAB-Map) for SLAM. As mentioned in section 3.4.3, RTAB-Map itself has been developed over the last decade by IntRoLab at Université de Sherbrooke in Canada. This section presents how RTAB-Map was configured for this robot. In this project, RTAB-Map was initially installed in the form of a released binary for ROS Indigo. Because of a

[5]https://github.com/googlesamples/android-BluetoothChat/...

crippling bug (ref. section C.3) which was fixed in the source code of `rtabmap_ros`, but not in the released binary, this package had to be built and installed from source.

4.9.1 Configuration

As all ROS programs which are a part of the robot system, RTAB-Map will run as a node that subscribes and publishes topics. The first task in configuring RTAB-Map is to connect the robots sensor data to the RTAB-Map node. The node, called `rtabmap`, can build 2D occupancy grids and/or 3D point cloud representations of the environment. In this project, RTAB-Map is configured to do both. The configuration is based on a guide[rta] provided by the developers. To perform SLAM, the mapping node subscribes to odometry, 2D laser scans and camera information. There are five possible sensor configurations with the Kinect[rta]:

1 - Kinect + LIDAR + Odometry

Sensor data can be sent directly to `rtabmap`.

2 - Kinect + Odometry + Fake 2D laser from Kinect

2D laser scans are generated by passing depth images from the Kinect through the node `depthimage_to_laserscan`.

3 - Kinect + LIDAR

This is the configuration that was used in this project. Odometry data is generated by a the scanmatcher within Hector SLAM (section 3.4.2).

4 - Kinect + Odometry

This configuration is suitable for uneven surfaces, i.e. when the vehicle is not constrained to a plane. Supports *roll*, *pitch* and *yaw* rotations.

5 - Kinect

In this mode, odometry will be generated by the `rgbd_odometry` node bundled with the `rtabmap_ros` package. This node publishes odometry messages based on feature correspondences in consecutive RGBD images received from the camera.

`rtabmap` subscribes to two image topics. One topic for depth images, `/openni/depth/image_raw`, and another for the rgb image, `/openni/rgb/image_raw`.

4.9.2 Adding 3D Obstruction Detection

The package `rtabmap_ros` contains a 3D obstacle detector in addition to the mapping node `rtabmap`. To enable 3D obstruction, point cloud data from the Kinect is passed through the nodelet `obstacles_detection`. The filtered point cloud is sent to `move_base` where it will be used in the local costmap.

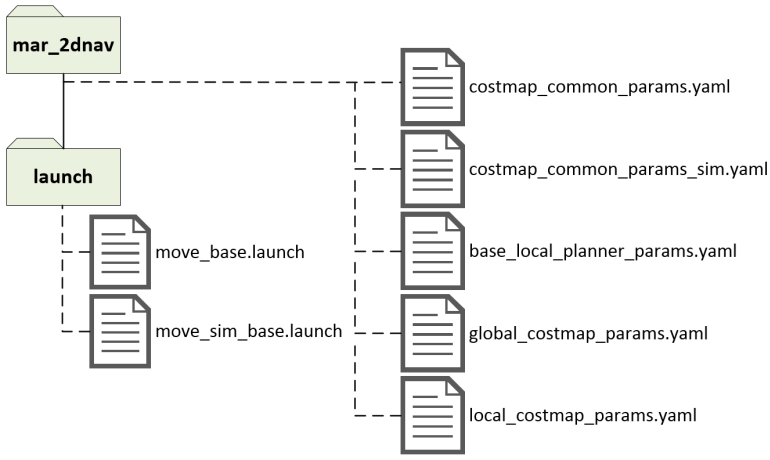


Figure 4.20: Files for configuring and launching the navigation stack.

4.10 Navigation

Autonomous navigation in this project is limited to use of the navigation stack in ROS, which was introduced in section 3.5. The stack will be used in its most basic form. The configuration was initially based on a tutorial at `ros.org`[6]. The implementation consists of one launch file for the node `move_base`, and a set of configuration files for each component within `move_base`. Figure 4.20 shows how these files were structured. As with RTAB-Map, the configuration process consist of connecting `move_base` to the rest of the network, by deciding which topics it shall subscribe to.

4.10.1 Local Planner Parameters

The configuration file for the local planner was configured with some initial settings before testing in the simulator and on the live system. The final parameters can be found in the file `base_local_planner_params.yaml`.

[6] http://wiki.ros.org/navigation/Tutorials/RobotSetup

4.10.2 Common Costmap Parameters

As the same costmap package is used for both the local and global costmaps, there are some configuration parameters that are common for these costmaps. The tunable parameters include the robot footprint and the costmap inflation radius.

Obstruction Detection

Configuration of the 3D obstruction detector, consists of linking a point cloud topic to the node `obstacles_detection`, and pass the resulting filtered pointcloud to `move_base`. The local costmap configuration file `local_costmap_params.yaml` must be configured to receive this point cloud by adding a `point_cloud_sensor` field to the file. The result of a successfully configured 3D obstacle detector is shown in figure 4.21 and 4.22.

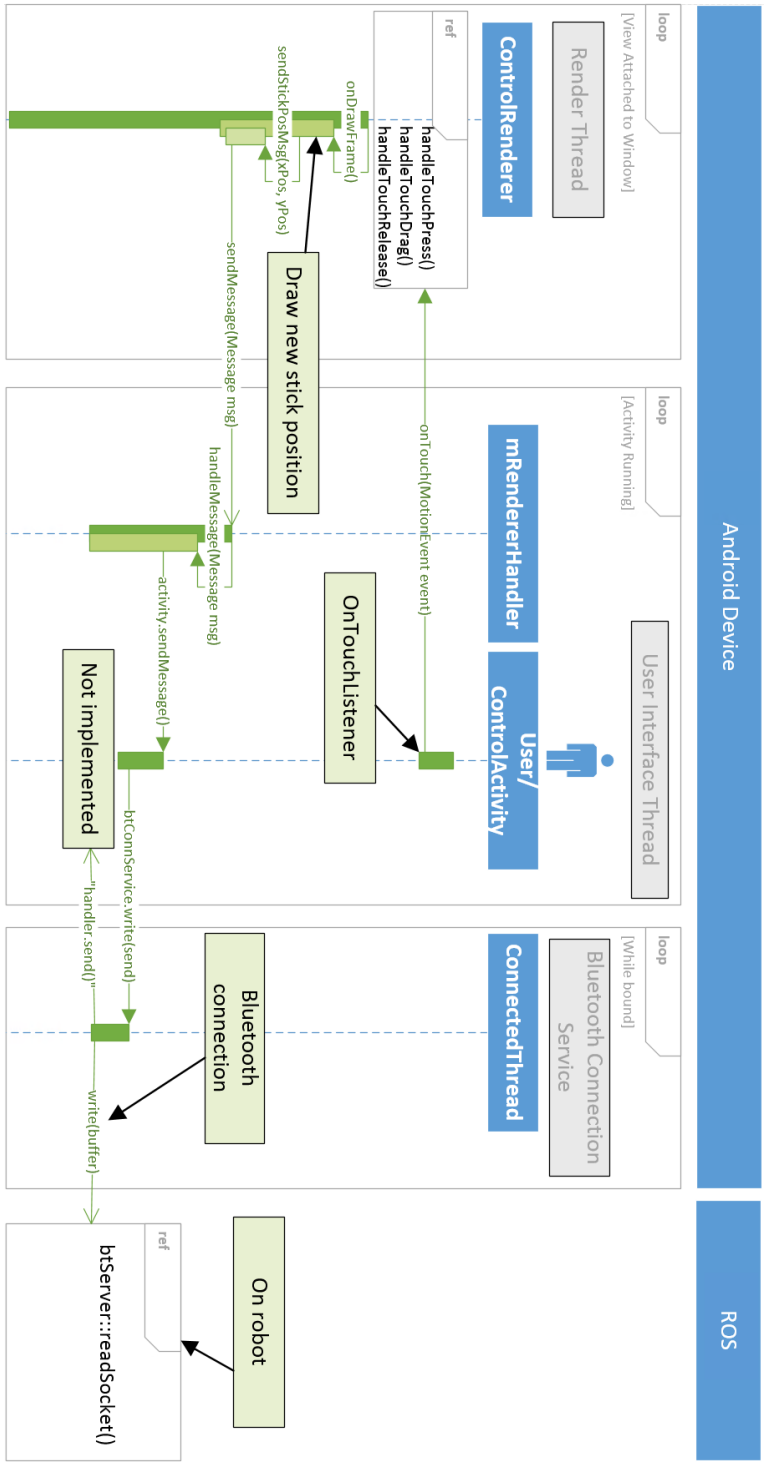
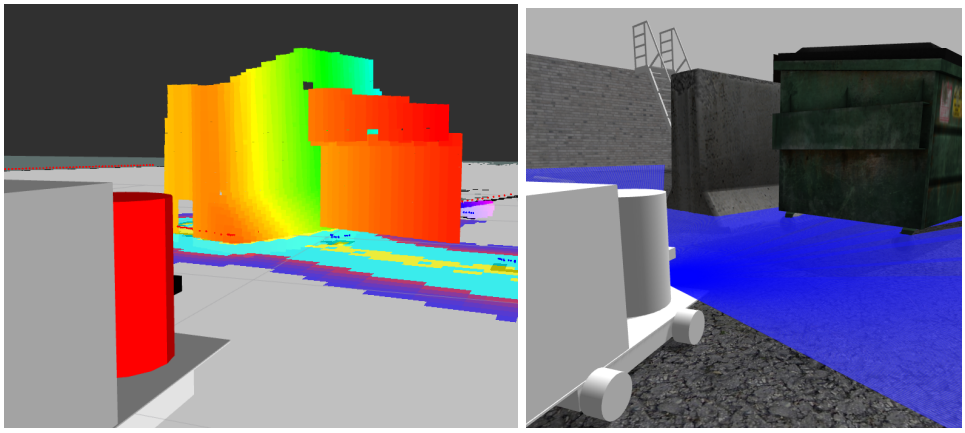


Figure 4.19: Sequence diagram illustrating how user touch gestures are detected and propagated through the application, before being transmitted as commands to the robot.



(a) A point cloud representation of the obstruction. Notice how the local costmap is based on the detected point cloud.

(b) The obstruction in the Gazebo simulator. Notice how the LIDAR only detects the wheels below the container.

Figure 4.21: Detecting obstructions in 3d.

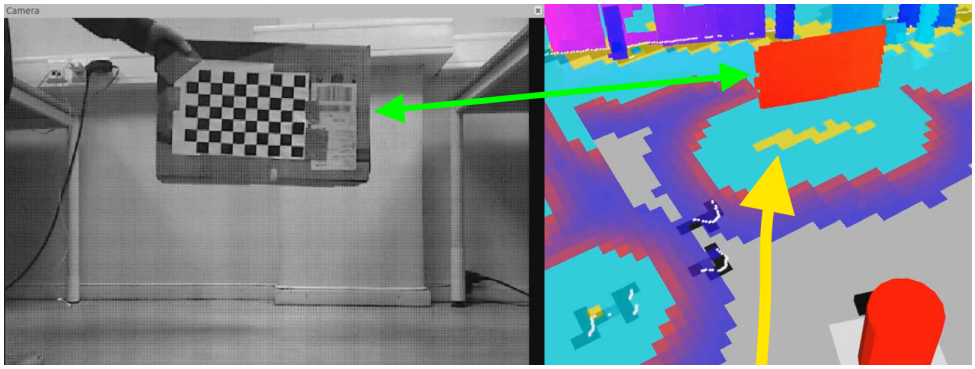


Figure 4.22: 3D Obstacle detection with the live robot. The nodelet `obstacles_detection` filters out the floor and publishes a point cloud which can be sent to the `move_base` node. The yellow arrow points to the local cost map, which is based on real-time sensor data and used by the local planner.

Chapter 5
Results

5.1 Introduction

This chapter presents how the robot and the supporting implementations were tested and the results that where obtained. The same software system was used for both the simulated and live robot. It was still necessary to have some separate launch and configuration files for the simulated model and real hardware version (section 4.3). Section 5.2 provides an overview of the various features and components that were tested, and how the tests were performed. Next, in section 5.3, a brief overview of the results are presented. The two last sections will go through the more nuanced results and events from the simulations and live trials respectively. It was difficult to come up with a rigorous test plan because of time constraints. Much of the testing was done together with parameter tuning.

5.2 Testplan

The tests listed in tables 5.1 and 5.2 will be carried out in the simulator as well as in the real world. They will mainly focus on the navigation stack and RTAB-Map.

Supporting Functionality

Evaluate	Description
Mobile application, "Robot Leash"	Use the mobile application to manually steer the robot.
Operator Control Station	Steer the robot from the OCS while monitoring the robot through the live video stream.
Motor controller on XMEGA A3BU	Verify ability to command the wheels. Confirm that the vehicle stops when velocity commands from ROS are absent.

Table 5.1: Supporting Functionality

Core Functionality

Evaluate	Description
Multi Session Mapping	Verify that the robot can rediscover areas which have been mapped in a previous mapping session.
Loop Closure Detection	As a core functionality in RTAB-Map, it is critical to evaluate the loop closure mechanism.
Autonomous Navigation	Perform a set of tests on the navigation stack. The tests should evaluate path planning with moving obstacles. Different parameters should be tested and evaluated. Observe how the robot handles narrow passages. Evaluate robustness of the navigation stack for this robot.

Table 5.2: Core Functionality

5.3 Brief Summary of All Results

Mobile application, "Robot Leash"

The mobile App works as expected. The user can establish a Bluetooth connection to the robot and send velocity commands to the real and simulated mobile base. This tool proved to be invaluable, and was used during all mapping sessions, both real and simulated. An emergency stop functionon in the Bluetooth server node, `bt_server`, was tested by switching of the Bluetooth adapter on the robot computer while driving. On the server, this event is successfully handled within the function `clientDisconnected()`.

Operator Control Station

Through the OCS, an operator can connect to the robot via a TCP socket, stream video from an URL and control the robot by using the mouse. The current OCS is a minimal application capable of demonstrating just the features mentioned, and not much more. The Kinect's narrow field of view makes it difficult to control the robot with high accuracy. A video demonstration of the OCS is available as a digital attachment to this report under the name "ocs_test".

Motor controller on XMEGA A3BU

The motor control card can receive velocity commands received from ROS, and translate them into wheel commands. The vehicle slows down, but does not come to a complete stop when the `motor_driver_interface` program in ROS is killed. The function `ovf_interrupt_callback` is intended to be responsible for stopping the

robot. Comments on this issue, and an untested suggestion for a possible solution is added to the source code. When this node is killed, the motor control card must be reset. The motor control card's ability to move the robot is demonstrated in all recorded videos of the live robot. When the robot system was running on internal battery power, the motor control card had a high probability of resetting. This is caused by poor Electromagnetic compatibility (EMC) due to noisy inverters and poor immunity in the motor control card. The problem can be mitigated to some degree by ensuring that the motor control card is properly grounded to the robot chassis.

Multi Session Mapping

The simulated robot struggled with false loop closures, which in turn lead to wrongfully merged maps. In Gazebo, the method is more vulnerable to similar features in different parts of the world. This was not a problem with the live robot system, which was able to successfully merge maps on all attempts. An example of multi session mapping was recorded to the video "live_multi_session_mapping" enclosed in the DVD.

Loop Closure Detection

Successful live loop closure detection is demonstrated in the video "live_mapping_succesful". The simulated environment was a bit more challenging, as demonstrated in the enclosed video "sim_wrong_loop_closure".

Autonomous Navigation

Autonomous navigation was evaluated based on the ability to reach a feasable goal state, and the ability to avoid static and dynamic obstacles. The global planner works as expected in both the simulator and in the real world. The simulated robot was frequently unable to reach its goal. The real robot successfully navigated a static and dynamic obstacle course (video "`live_navigation_1`" and `2`, and "Obstruction detection and avoidance"). It would fail if the new obstructions that come within the robots field of view are too close to the sensors, i.e. closer than $\approx 0.6m$. This can occur when the robot rounds tight corners or when a person steps in front of the robot within the minimum range. Distance margins to obstacles were sometimes too small. In some situations, the robot would turn back toward a detected obstacle before the path was clear.

5.4 Simulation Results

The system was tested on a simulated model of the robot in Gazebo. A simulated environment, `Asphalt.world` shown in figure 5.1, was populated with objects and

clutter in order to provide a test environment with distinctive visual features for the visual mapping approach, and obstacles for the navigation stack.

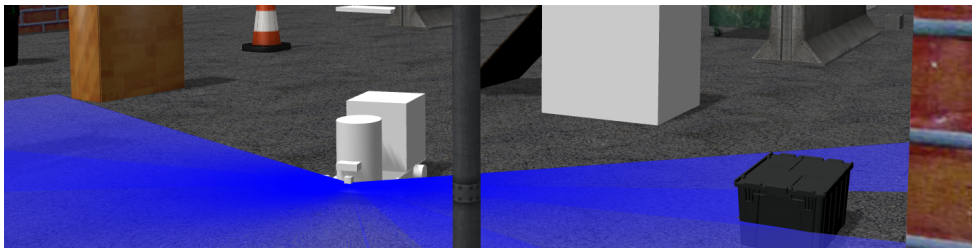


Figure 5.1: The ”Asphalt” world in Gazebo.

5.4.1 Mapping

RTAB-Map on the robot in Gazebo allowed controlled testing of edge cases in a controlled environment. The ”Asphalt” world proved to be a challenge for RTAB-Map - at least with the parameters that were used during testing. Figure 5.2 shows an example of a resulting 3D map.

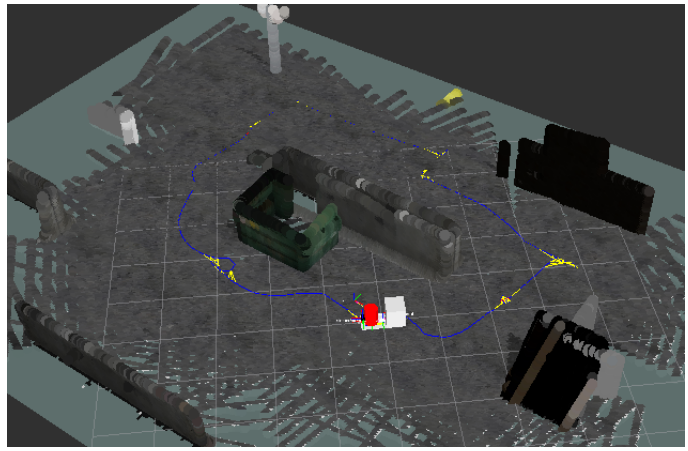


Figure 5.2: An example of a resulting point cloud map after running RTAB-Map in Gazebo.

The map quality varied greatly between the mapping trials. The path of the robot was found to have a significant impact on the recorded path between the stored locations in the RTAB-Map system. An example of a problematic path is to drive the robot in parallel to a wall. Such a path creates few or no distinctive features as long as the robot follows this path. Figure 5.3 shows two problematic events. First, a loop closure is detected based on similarities on the asphalt plane. Because the

loop closure is wrong, an incorrect pose transform correction will be propagated backwards along the path of the robot, ultimately resulting in a displaced map. In these situations it is of no help that RTAB-Map stores the depth of each feature.

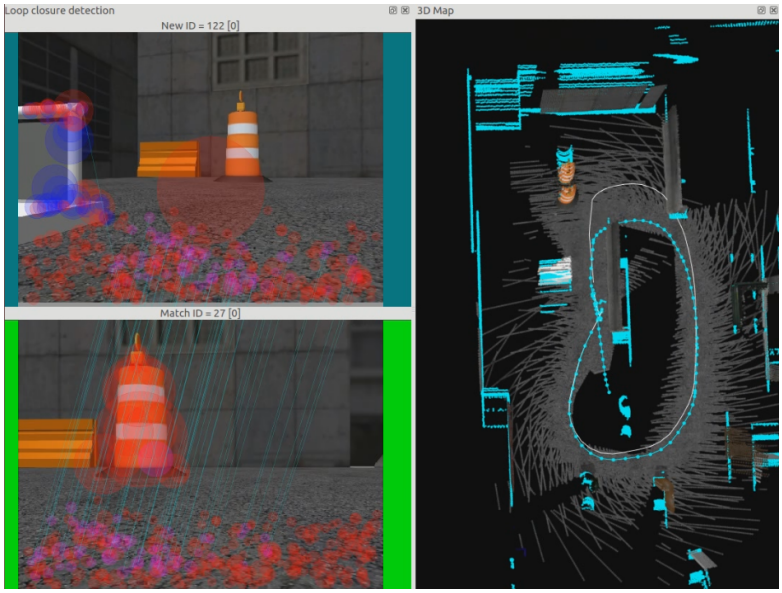


Figure 5.3: Example of an incorrect loop closure detection. The pink circles indicate matching features. The right part shows an incorrect map adjustment. Observe how the matching features are located on the asphalt plane.

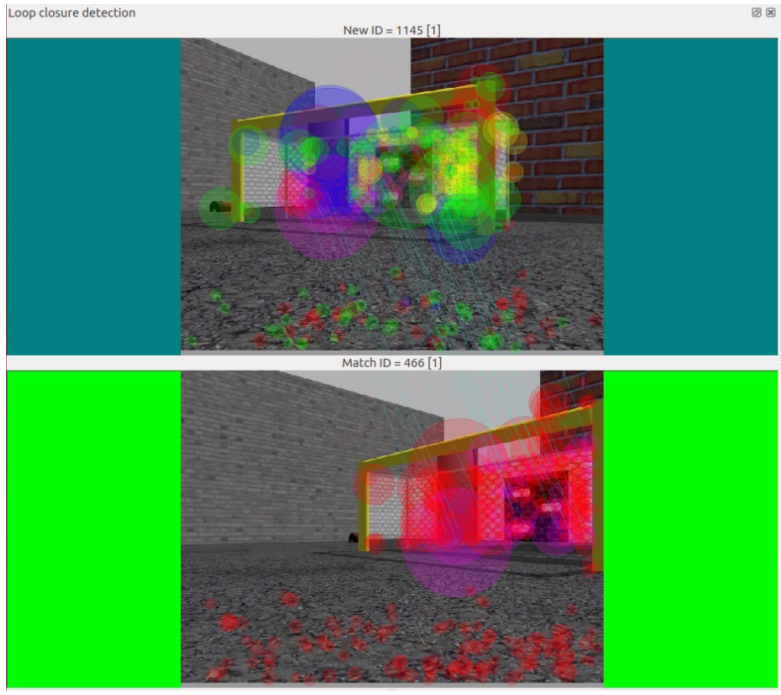


Figure 5.4: An example of an accepted and correct loop closure hypothesis. This example is from the ”Asphalt” world simulated in Gazebo.

Another error occurred during a multi session mapping trial. The robot detected a wrong loop closure at a location with similar features to a previously mapped location. This resulted in the map seen in figure 5.5. Other localization problems were apparent when driving in open areas. The estimated location of the robot would fluctuate when changing the robot's heading as features passed in or out of view.

Sensor settings is another factor that had a high impact on the SLAM quality. When sensor ranges of the Gazebo sensor plugins are set according to the sensors technical specifications, both localization and mapping will struggle. An increased sensor range in the simulated sensors would increase the SLAM robustness.

5.4.2 Autonomous Navigation

Testing the navigation stack in Gazebo fulfilled two goals. The first goal was to learn how the system behaved with different parameters and to uncover potential problems with the system, before attempting to test the live robot. The second goal was to evaluate the ability to relocate the robot and avoid obstacles.

Figure 5.5: An example of incorrect map merging. This case occurred in the "Asphalt" world simulated in Gazebo.

The for the first trial, the robot footprint was extended well beyond the physical mobile base. The purpose of this was to prevent obstacles from getting too close to the Kinect and LIDAR. As the minimum detectable range for the Kinect is $0.5m$ (measured minimum depth), the footprint was extended $0.5m$ beyond the front of the mobile base. A second parameter to be tuned is the obstacle inflation radius, i.e. the radius beyond each obstacle that is expensive or impossible to traverse. Figure 5.6 illustrates both the big footprint and the obstacle inflation radius.

During trials with the big footprint, the robot showed good collision avoidance capabilities but a reduced ability to follow the planned path as well as an aversion to narrow passages. Setting the obstacle inflation radius is also a dilemma in choosing between large margins for the global path or the ability to navigate through narrow passages.

In later navigation sessions, the robot footprint was reduced to a size slightly larger than the physical robot base. During some of the testing sessions, the robot would get stuck near obstacles.

Sometimes, the robot would never actually reach the goal state, but rather circle around it. This problem was not solved, but more relaxed goal tolerances mitigated the issue a little.

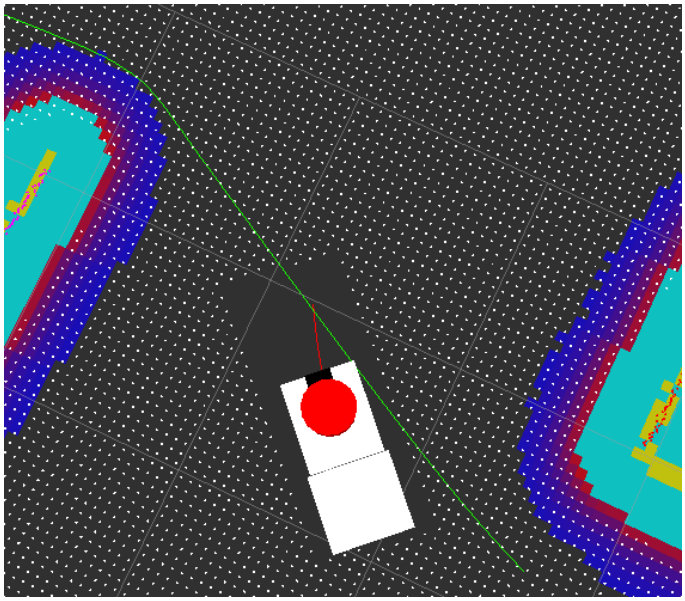


Figure 5.6: The robot footprint is illustrated by the clear rectangle that surrounds the robot model. The coloured areas are map locations with high cost.

5.5 Live Robot Results

5.5.1 Mapping

Due to time constraints, it was no time to tune the parameters of RTAB-Map. RTAB-Map is therefore used with the default parameters. An important distinction between the real and simulated system is found in the actuator, i.e. the motor controller of the mobile base. In the simulated system, the motor control card is emulated by a skid steering plugin. While the exact functionality of the skid steering plugin is unknown, it does ensure that the wheels follow the linear and angular velocity commands provided by ROS. This is not the case for the real motor control card, as it lacks a feedback loop. The real robot velocity was in general slower that its simulated version.

Loop Closure Detection

Loop closure detection was carried out on the first floor of Gamle Elektro at Gløshaugen, NTNU (figure 5.7a). This environment provides a good mix of featureless and feature rich surroundings. It will also provide an additional challenge because of the many students that use the hallways. Most importantly, there are loops in the environment that allow testing of vision based loop closures and odometry error correction.

The first large-scale test run revealed two problems with the implementation, the first being odometry errors and the second being a failure to visually detect loop closures. Odometry based on laser scans would wrongfully indicate a change in the robot heading in some cases, and in other cases fail to correctly indicate heading changes when rounding corners. The test run was recorded, and is available as a video on the DVD by the name: `live_first_large_scale_mapping.ogv`.

In later experiments it was found that different mapping techniques and path choices could either prevent or cause odometry errors. It was also found that it is helpful to start a mapping session in an area that is rich in distinctive features. The map and floor plan comparison shown in figure 5.7 shows a map where a loop closure was successfully detected. In the same figure, notice how the upper hallway is misaligned to the rest of the map. The robot failed to detect any good visual features in this area. Figure 5.8 shows the resulting point cloud map of the same area.

In figure 5.8, notice how the detected features are located above the floor plane. This is a major difference from the simulated trials where a significant amount og the detected features were located at the ground plane, as indicated in figure 5.3 and 5.4.

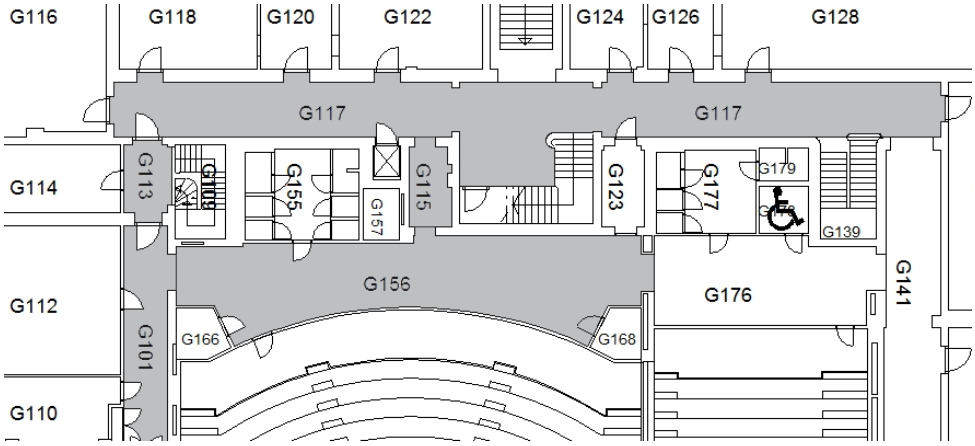
Multi Session Mapping

Multi session mapping was tested by performing an initial mapping run and then starting a second mapping session from an unknown(for the robot) location. All multi session mapping trials were successful. No special effort was made to reproduce the erroneous map merge(figure 5.5) that occurred in the simulator.

5.5.2 Navigation

Initial navigation sessions were dedicated to tuning the navigation stack. Changes were made to the parameters of the local planner. The minimum speed was increased to overcome friction. The goal tolerance was increased up to $20cm$ xy-tolerance and $0.20\,radians$ yaw tolerance.

(a) Resulting occupancy grid after a mapping session. The mapping method is struggling with the hallway in the upper part of the image.



(b) Floor plan of Gamle Elektro, first floor.

Figure 5.7: Comparison between mapped occupancy grid and floor plan.

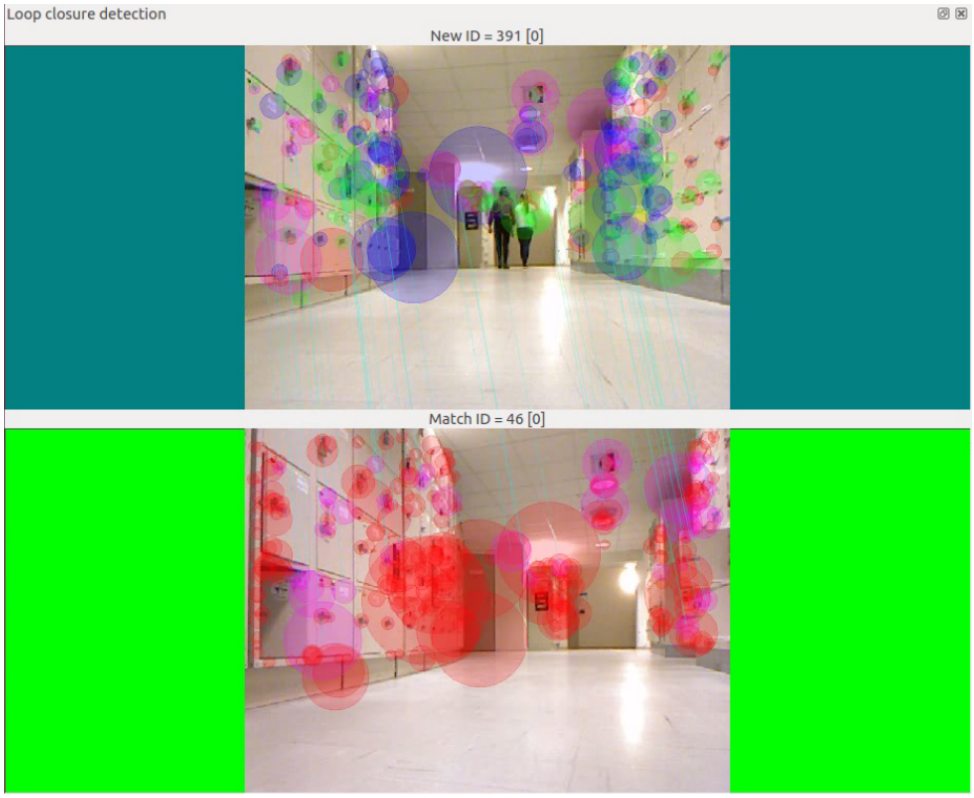


Figure 5.8: An example of an accepted loop closure hypothesis during a live mapping session. As before, the matched features are indicated by the pink circles.

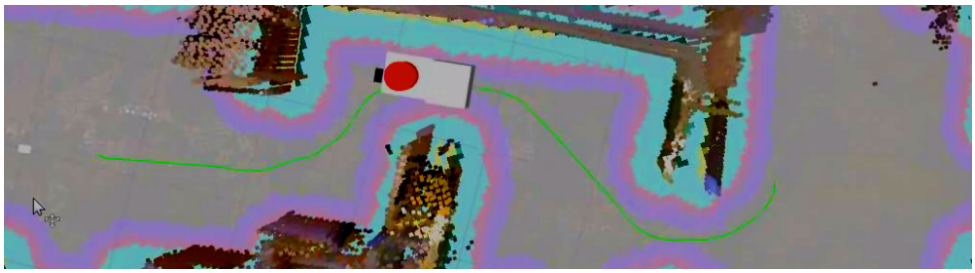


Figure 5.10: Global planning with the live robot. The green line illustrates the globally planned path. The inflated obstructions in the global costmap are highlighted as colored spots.

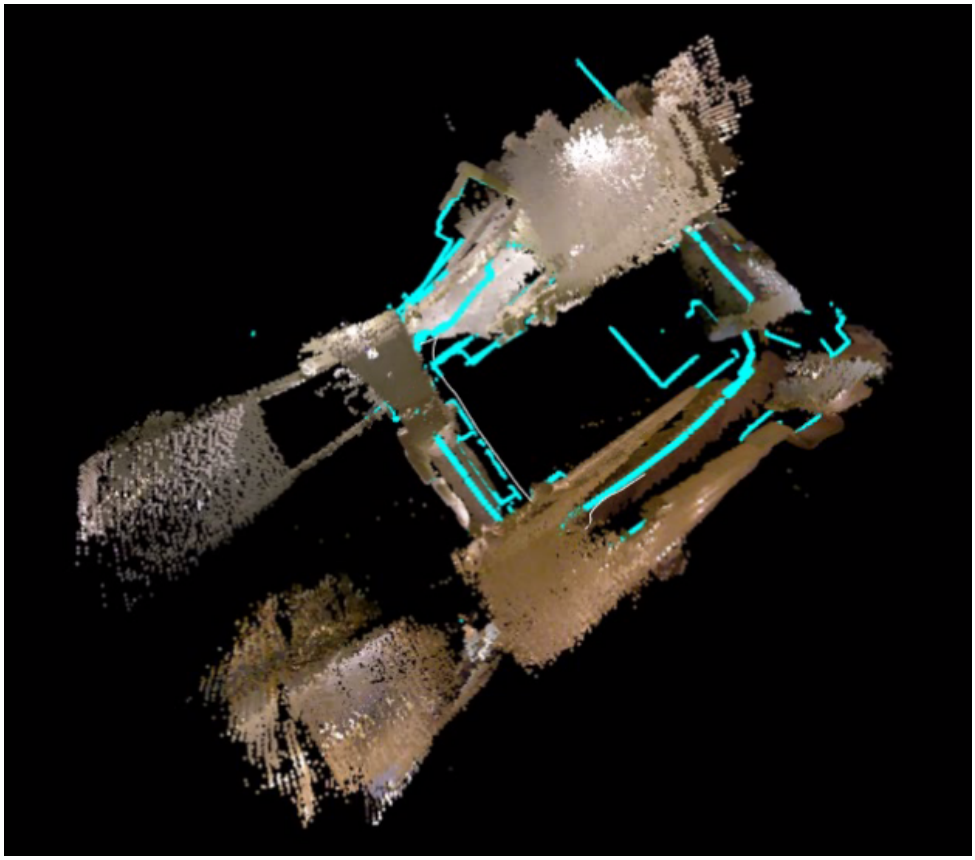


Figure 5.9: The resulting 3D map of the same area as in figure 5.7a.

Navigating Among Static Obstacles

The first live navigation trials were performed by setting up a static obstacle course in a hallway. Only a few people were using this hallway, so multiple trials could be carried in under similar conditions. Most navigation trials were carried out in mapped areas. The robot performed well when navigating in known environments. In unknown environments, the robot will still plan an optimistic path, but if it is unable to map or detect new obstructions, it will collide. An example of a typical test drive is shown in figure 5.10.

Avoiding Moving Obstacles

The navigation stack is configured to use a static global map for global planning and a local cost map that is linked with the mobile base and is based on real time sensor data. The local cost map will receive LIDAR laser scans, Kinect point clouds and point

clouds representing obstacles detected by `rtabmap`'s nodelet, `obstacles_detection`. Being a real-time map, the local cost map should enable the local planner to avoid people and other non-static obstacles.

Moving obstacle avoidance testing was performed by having a person move into the planned path of the robot at different distances. Other avoidance situations would occur randomly as people were walking by the robot in the hallways. Tests showed that the robot is able to avoid non-static obstacles if the obstacle is observed at a distance larger than $0.5 - 0.8m$. Figure 5.12 shows that the robot has successfully planned a new path around a person. If an obstacle appeared any closer than this, the new circumnavigating plan would either be too close to the original plan or not be planned at all. Detection and planning was not instantaneous. Some time would pass before the obstacle was detected and a new local plan was generated. This detection delay reduced the detectability of people walking by the robot. Figure 5.11 shows an example of when the local cost map was lagging behind the actual moving obstacle.

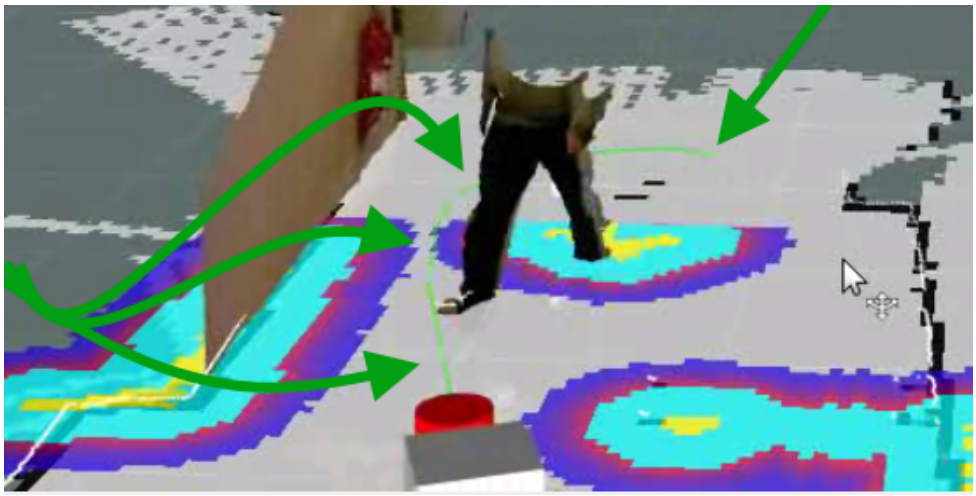
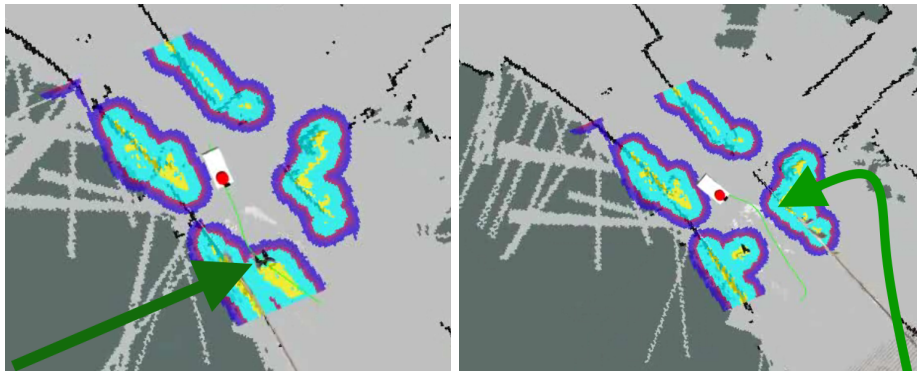


Figure 5.11: Avoiding moving obstacles with a new plan that circumnavigates the detected obstruction. In this situation, the obstacle was moving too fast for the local planner. The right leg is not yet registered as an obstacle.



(a) A person has moved into the path of the robot.

(b) A new path is planned, avoiding the new obstacle.

Figure 5.12: Moving obstacle avoidance. The local cost map, shown as coloured spots on the occupancy grid, is based on real-time sensor data.

Chapter 6
Discussion

6.1 Introduction

This chapter will assess the implementations presented in chapter 4 based on their performance which is detailed in chapter 5. The following discussions will qualitatively evaluate how suitable the implementations are for offshore robotic maintenance. The first section will discuss the performance of the system as a whole. Next follows a discussion on the strengths and weaknesses of the navigation and mapping systems. The chapter is concluded with a section on future work.

6.2 Overall Assessment

The overall system, as it is at the end of this master's project, is a functioning proof of concept for a mobile autonomous robot. All planned modules were implemented. Some of these, however, are only capable of demonstrating basic functionality and the possibility of more robust and complete implementations. The most important features, camera based mapping and navigation, were successfully implemented and configured. The current implementations leaves much room for further improvements in terms of robustness, parameter tuning, functionality and ease of use.

6.2.1 Choice of Development Tools

Using ROS as a development framework might have been the most important factor that contributed to a functional solution. In the end, ROS proved to be a flexible and rich tool, despite its novel structure and initial learning curve. Experienced users of ROS will most likely be able to rapidly implement and test robot concepts. Since its inception in 2007, ROS has become a mature and rich set of tools and functionality packages that anyone can implement and develop further. Further development by users is even encouraged by many of the ROS package creators. The node structure in ROS is also a good way of structuring the entire system into self contained, manageable and reusable modules. This makes it easier to reuse parts of

this implementation in later projects, and will hopefully benefit ensuing projects on this topic.

6.2.2 Assessment of Prototype Design

Roughly estimated, the robot prototype have travelled a distance of approximately $1km$ during the master's project. Of these, roughly $100 - 200m$ were driven in autonomous mode. The bulk of these distances were accomplished over a period of 3 intensive days, while the rest was spread throughout the last third of the semester. Time available for testing was somewhat reduced as a consequence of breakdowns or depleted batteries. Another cause of the delays are caused by EMC issues between the motor control card (A3BU) and the AC inverters. It has not been made clear if the problem is caused by only one or both of the inverters. It was possible to increase the immunity of the motor control card by grounding it to the robot chassis. The current grounding connection is not reliable at this time.

So far, the robot prototype has been developed for a few specific functionalities, and coexistence between the robot systems has not been a priority. Future projects could benefit from a comprehensive and long-term approach towards designing a maintenance robot. The following paragraphs will go through some specific design choices on the current robot, and discuss potential shortcomings.

Suboptimal Power Supply

Some time was spent on building a new rear compartment to house the various robot equipment, including the battery and both on-board computers. For internal power, the robot uses a 12 V car battery as a power source. Figure 4.2 and 4.4 highlights the inefficient power supply layout. The Kinect requires a supply of 12V DC, which is supplied from an adapter plugged into a 230V AC source. Receiving 12 V directly from the battery might be a more efficient solution.

Current Design and Robotic Maintenance

A typical offshore installation floor will most likely be made of steel and steel gratings with many holes, gaps and sharp edges[GP08]. The current mobile base is better suited for completely even surfaces, and would benefit greatly for a more rugged set of wheels. [GP08] is also referring to a minimum size for passage ways that could serve as guidelines for later prototypes. The suggested minimum workspace boundaries, $0.75m$ wide and $1.5m$ high corridors, imply that the prototype used in this project is too big, given its footprint of $80cm \times 37cm$. Another problem is that the Kinect would have trouble with mapping and navigating in this environment, given its measured minimum range of $0.5m$, not to mention the reliable minimum depth of $0.8m$. In this author's specialization project[Lin15], the minimum measurable depth

of the obstruction detection was found to be $\approx 40cm$. The minimum range was limited by the maximum disparity parameter. Increasing the detectable maximum disparity could reduce the detectable range even further, thus making it a better option for narrow spaces where depth ranges are short.

As the motor control card is an open loop system, slopes and increased friction for whatever reason will affect the speed of the mobile base. This may cause the base to slow down, speed up or even stop completely if it is driving up a slope that is too steep.

Given that the navigation stack in ROS is thoroughly tested on square or circular bases, designing a new base to be either square or circular could increase the robustness of the navigation system. A holonomic drive could also make the robot more maneuverable, which may be useful in tight spaces.

The current design of the mobile base is struggling to support the weight of the robot. The small wheels are struggling with small obstacles such as door thresholds or floor gaps in the entrance to elevators. MIMROex, a comparable robot, is equipped with a variant of a differential drive base with larger wheels suitable for driving over steel gratings and slippery surfaces.

6.2.3 Success and Quality of the ROS Integration

Integrating and configuring ROS with the existing robot was the most time consuming task of this project. The current implementation is capable of autonomous navigation and long-term map building. Remote operation from the OCS and Android device is also possible. A major shortcoming of the current configuration is the lack of usability. The current system is cumbersome and difficult to use. A potential difficulty may be to separate the parts that work well from the more unstable parts of the system.

6.3 Assessment of RTAB-Map

The mapping session results demonstrated both strengths and weaknesses in the chosen mapping method, RTAB-Map. The results show that multi session mapping works rather well if the conditions are favorable, e.g. in environments with a sufficient amount of detectable visual features.

6.3.1 Quality and Thoroughness of the Tests

RTAB-Map was tested in a diverse set of indoor environments as well as in the simulated ”Asphalt” world in Gazebo. A significant shortcoming of these tests was the lack of testing and comparison of different parameter settings for the method.

Further parameter tuning and better mapping techniques could have benefited the mapping performance.

Another shortcoming is the small number of live mapping trials. Configuring and learning to use the mapping system was a time consuming process. Problems with the robot hardware and the environment itself gave rise to additional delays. The laptop running ROS and the on-board car battery had to be recharged periodically, which took a considerable amount of time. The live loop closure tests were carried out at times when a lot of students were moving through the hallways. These factors made it difficult to perform comparable tests, and proved to be a complicating factor for the appearance based mapping system.

A third weakness in these trials is that the system was tested with only one sensor configuration. RTAB-Map can utilize both a RGB-D camera, a LIDAR and odometers, but these tests were only carried out with the Kinect and the LIDAR.

Simulation in Gazebo has been an invaluable option throughout the project. However, the inherent difference between the real and the simulated world should be discussed. A poorly designed simulator world could call into question whether the simulated results are representative for the mapping performance. A notable difference between the real and simulated trials, which should be kept in mind when reading the next subsection, is the how feature rich the simulated floors are. This will become apparent when the reader compares how the feature detectors extracts features. In the recorded material bundled with this thesis, it is apparent that the simulated floor is very feature rich compared to the real world floor. In retrospect, a simulated world with a simpler floor texture might have served this project better than `asphalt.world`.

The developers of RTAB-Map have created several datasets which can be used to reproduce their results and tune the mapping system. One such dataset, from the computer game "Need for Speed: Most Wanted", shows that RTAB-Map does work in a simulated environment[1].

6.3.2 Weaknesses

Appearance based loop closure detection with RTAB-Map has many confirmed and potential weaknesses that must be addressed. Figure 5.3 from a simulator session illustrates an incorrect loop closure detection with a subsequent incorrect odometry correction of the previously visited locations. The figure shows that the matched features are based on the ground plane in the simulated world. Having a feature rich ground plane could be a weakness with the simulated world, as it is not was not a good analogue to the real world. The depth map generated by the Kinect was in fact

[1] RTAB-Map: NFSMW data set (part 1): https://www.youtube.com/watch?v=kghs6XM8Yzw

quite sparse at the ground plane. Another error that occurred during simulations is incorrect merging of two maps of the same area (figure 5.5). This particular event was caused by having two very similar locations in the same area.

A potential problem is that the appearance of the environment will change based on the time of day, time of year and potential wear and tear on the surroundings. How robust the feature detectors (SIFT, SURF, Oriented FAST and Rotated BRIEF (ORB) etc.) are to such changes was *not* investigated during this project.

6.3.3 Strengths

RTAB-Map is packed with features, parameters and useful tools. It supports many sensor configurations, including stereo cameras. The ROS wrapper makes it easy to integrate the method into an existing robot system. The developer or user has access to hundreds of parameters to tailor and fine-tune the mapping system. Object recognition and 3D obstacle detection is also useful features that will support a maintenance robot.

Another strength is that RTAB-Map works with any RGB-D sensor. It might be possible to to use both an active camera and a stereo camera, and switch between the two, based on the conditions. A stereo camera is better suited for outdoor mapping than a Kinect, due to the presence of IR wavelengths in sunlight.

6.3.4 Suitability For Robotic Maintenance

RTAB-Map has a wealth of configuration options with many capabilities that can make it suitable for a remotely operated and autonomous maintenance robot. The built-in object recognition capabilities are useful. Unfortunately, the object recognizer was not tested.

As mentioned in section 6.3.2, it is clear that an appearance based method may be vulnerable to natural variations in the environment. This fact could completely rule out RTAB-Map as an outdoor SLAM system. The suitability of the method will also depend on the robots intended task. Based on the discussion so far, it may seem that RTAB-Map can be useful for an indoor robot during normal operation of the facility.

Section 2.2 discussed the hazards of offshore O&G production and how a robot could respond to crisis situations such as hydrocarbon leaks, toxic gases or fires. Such events could alter the appearance of the environment, which in turn may knock out the SLAM capabilities a robot.

6.4 Navigation

Integrating the ROS navigation stack into a new mobile base was in itself a fairly simple procedure. Finding a good configuration turned out to be a more complicated process. Navigation was tested on both the simulator and the live robot. It became apparent during the testing sessions that the behaviour of the simulated robot was not analogue to the real robot. Recall that the simulated robot is controlled by a slip steering plugin in Gazebo, while the real robot is closer to a differential drive vehicle. In addition, the motor control in the real robot is an open loop system. The wheel commands from ROS will normally result in a lower linear velocity and yaw rate. A consequence of this discrepancy between the real and simulated robots, is that the performance assessment of navigation stack in the simulator will have a reduced weight.

6.4.1 The Tuning Process

There are no official tuning strategies for the navigation stack in ROS besides a basic guide. The guide serves to give users a general idea of where to start and what to check[ROSg]. The tuning process is currently a ”change and check” process, partially based on guesswork or of copying similar solutions from other projects. This is far from ideal, as it is both time consuming and a hindrance for finding an optimal solution.

It should be noted that the navigation stack has been thoroughly tested on robots with square or circular bases. The highly rectangular base ($80cm \times 37cm$) on this robot may have been a handicap.

As with the SLAM test sessions on the live robot, the cumbersome hardware and limited battery life significantly constrained the amount of time available for testing of the navigation stack.

6.4.2 Performance

Live testing showed promising results, despite a few quirks. The live robot would reliably plan a path to a goal location and move the base to this location, given that the goal was feasible. Sometimes, the robot would stop a few centimetres before reaching the goal location.

Based on the test results, the live robot should have a higher minimum speed setting than the simulated robot in order to overcome friction and other resisting forces. The reason for this is that the velocity command is matched against wheel speeds when the robot is off the ground, i.e. the wheels are spinning freely. If the motor control card is expanded with a speed regulator, the minimum speed may be reduced.

6.5 Various Topics

6.5.1 The Kinect

As mentioned in section 3.3.2, the Kinect is an active sensor that measures depth by projecting an infra red speckle pattern onto the surroundings. Kinect for Xbox 360 can't be used in commercial applications because of its license. There are, however, similar sensors that can replace the Kinect.

6.5.2 Open Source Software and Security

ROS and other open source projects thrive on active communities of contributors. Both PCL and ROS, as well as many other libraries and frameworks, are built on a collaborative effort from researchers and developers across the globe. This is open structure is great for speeding up innovation. Issues and bugs can also be discovered more quickly by anyone. Another benefit is that every detail in an open source project is open for scrutiny by those who want to use it. This is also a problem in terms of security. While anyone can find bugs and issues, the code is also open to those who are looking for possible exploits and vulnerabilities. If a system is targeted for sabotage, and it is widely known that the system uses open source software, it might be more vulnerable to security threats.

6.6 Future Work

6.6.1 Continued Work on This Project

There is much left to do on the project, even if the current robot system is used. The only hardware requirement to pick up the work where this author left off, is a new on-board computer running ROS Indigo. The mobile base will also require an overhaul.

Further Testing and Assessment of RTAB-Map

A more complete assessment of RTAB-Map may support a better discussion on its suitability for offshore maintenance missions. As mentioned in section 6.3.1, only one of the possible sensor configurations were tested. It may be that the mapping process will benefit from dead reckoning odometry from encoder wheels. Furthermore, since RTAB-Map can be used with stereo cameras, is could be useful to compare passive and active depth sensors.

Improve the Communication Protocols

Communication between ROS and the XMEGA A3BU, the Bluetooth device and the OCS, all use the same pattern: A start byte `":"`, the message with the speed setting

and a stop byte `"Esc"`. In later projects, it could be beneficial to implement a more robust and rich communication protocol with more options for remote operation. A first step may be to stuff all the bytes into a buffer prior to writing the message.

There is a problem with the dead-man switch implementation on the motor control card. A timer overflow interrupt is intended to trigger when new velocity commands seize to arrive, or if a deadlock occurs in the receiver cycle. When the interrupt is triggered, the motors will slow down, but not stop completely. This issue should be fixed in future projects.

Implement a Fully Functional Operator Control Station

At the end of this project, the OCS provided functionality for moving the robot, and displaying live video from the Kinect. Future projects could focus on development of new control station designs, or perhaps even integration with a VR telepresence system. Another option which may ease the implementation process, would be to distribute the ROS system over multiple computers (which is supported by ROS). A benefit of this is easy easy access to all ROS tools and features. A drawback is that SDKs for various VR equipment may be incompatible with Linux Ubuntu.

An important functionality on the server side, is to set velocity commands to zero when the connection to the OCS is lost, or when new velocity updates are absent. This functionality was not implemented.

Closed Loop Wheel Control

By implementing a closed loop wheel controller, the actual velocity of the robot will be closer to the commanded velocity. In the current system, a velocity command will be translated directly into a wheel speed setting.

6.6.2 Hardware

Several hardware-related issues became apparent over the course of the project - especially toward the final weeks. These issues are likely the results of many disconnected projects on the same hardware over the years.

Kinect Sensor Location

This is the first semester in which a Kinect has been used on the robot. At the moment, the sensor is placed directly over the LIDAR device at the front of the robot. Because the depth sensor in the Kinect for XBOX 360 has a minimum range of roughly $0.5m$, it cannot detect objects within reach of the robot arm. It is recommended to find a new location further back on the robot.

Combine Stereo Cameras with Kinect-like Sensors

As mentioned, both active and passive depth cameras have limitations. The Kinect does function in direct sunlight, but it can measure depth in the dark because of the projected infra red pattern. Passive depth sensors, for example stereo cameras, does depend on visible light to sense anything at all. While RTAB-Map do depend on visibility for loop closure detection, there are other SLAM methods, e.g. *Kinect Fusion*, which do not. An implementation could use a light sensor to sense light that may interfere with the Kinect. Light levels could be compared to a threshold and switch between the stereo cameras or the Kinect depending on how well each sensor will work in the current conditions.

On-board Computer Suitable for Moving Platforms

Because this author used his own computer to control the robot, all features related to ROS was removed from the robot at the end of the project. A new computer should be equipped with Solid State Drive (SSD) storage, as it is less vulnerable to vibrations and potential crashes.

Mobile Base

There were mainly two issues with the omni-wheels this semester: They are worn out, and one wheel slipped out of the motor drive shaft. The rubber on a few of the perpendicular rollers is loose and about to fall off the plastic rims. This causes the robot to shake, which can damage spinning hard disk drives or shake the sensors out of their calibrated positions.

Figure 6.1: Worn omniwheel

A new set of wheels should be able to carry the weight of the robot, and enable the robot to drive over small barriers such as door sills and steel grates. This will most likely prompt a redesign of the mobile base.

A new design of the mobile base should consider the potential benefits of a holonomic drive system.

6.6.3 Suggestions and Ideas

Autonomous Non-Destructive Testing

Advancements in Artificial Intelligence (AI), big data and machine learning opens up exciting possibilities for autonomous NDT. Branches of this technology is usually encountered in the context of image recognition, i.e. teaching machines to understand what they see. The same concepts may be applied to forms of NDT besides regular visual sensor input, such as ultra sound or eddy currents for corrosion detection.

Augumented Reality and Large Scale Kinect Fusion - Kintinuous

Kinect Fusion has great potential for augmented reality. Augmented reality is a concept which blends the real and virtual environment. This opens up opportunities to create realistic and immersive training scenarios for the operators. Unfortunately, Kinect Fusion is limited reconstructing a rather small volume depending on the resolution. By varying the resolution, volumes can at the least cover a normal office desk and at the most cover a room smaller than $\leq 7m^3$ [NIH+11].

Kintinuous is an experimental extension of Kinect Fusion for large scale volume reconstruction. A guide on how to build Kintinous can be found on the project's Github page[2]. The procedure is complicated, as it usually is for experimental builds. It is recommended to attempt the procedure on a fresh install of Ubuntu 14.04 or 15.04 [Kine].

There may be other 3D reconstruction technologies available today that are better options than Kintinuous. It is recommended to perform a thorough literature study on the topic before selecting a technology to work with.

[2] https://github.com/mp3guy/Kintinuous

Chapter 7
Conclusion

7.1 Problem Description Fulfillment

Point 1 - Theory and state-of-the-art Solutions

Chapter 2 presented some recent projects on robotic maintenance and inspection. Both MIMROex and Sensabot resembles the prototype used in this project. Of these two, MIMROex might be the best source of inspiration when considering new designs and implementations. Industrial maintenance robots are currently capable of teleoperated inspection, and autonomous manipulation of the processes are heavily researched.

Point 2 - Selection of Development Tools

The chosen tools and frameworks are ROS for the robot software and Qt for the OCS. ROS is introduced in chapter 3.2. These tools proved to be suitable for the task at hand. An Android device was used as a supporting tool during testing.

Point 3 - Test Platform

The mobile robot platform used in [Asp13], [Ber13] and [Lin15] was used in this project as well. A Kinect sensor and an additional on-board computer was added to the robot. A new shelf structure was placed on the robot in order to accommodate the new equipment.

Point 4 - Solutions and Implementations

Autonomous navigation, localization and mapping is among the fundamental tasks an autonomous robot will face. Visual SLAM capabilities were implemented by using RTAB-Map, which is presented in section 3.4.3. For navigation, the robot uses the ROS navigation stack, which is introduced in section 3.5. The navigation system receives obstacle information a depth camera and a LIDAR.

Point 5 - Assessment

An assessment of the project results is provided in chapter 6.

7.2 Final Conclusion

Project Objective

The main objective of this project was to implement a vision based solution to a problem faced by a mobile, autonomous maintenance robot. SLAM was chosen as the problem to be solved, as it is one of the fundamental requirements for a mobile robot. To meet the objective, the robot was configured to use ROS together with RTAB-Map; a system for large scale appearance based SLAM, developed by IntRoLab. An additional major implementation goal was to achieve autonomous navigation. A third objective was to implement an operator control station (OCS) where an operator can monitor and control the robot via a wireless connection.

Implementations

RTAB-Map is configured to use a Kinect for Xbox 360 in combination with a LIDAR to generate 2D occupancy grid maps and 3D point cloud maps of the environments. The current implementation is capable of building maps over multiple sessions. Scan matching from Hector SLAM provides odometry to RTAB-Map.

The robot has been configured to use the navigation stack in ROS. The navigation stack configuration enables the robot to plan and follow a path to a simple feasible goal. Additional capabilities include dynamic replanning in the case of obstructions, and 3D obstruction detection based on point clouds.

Three sources of control inputs, besides the navigation stack, was implemented: keyboard inputs for simple testing and input mode settings, commands over a TCP socket from the operator control station (OCS) and commands received via a Bluetooth connection. The OCS is capable of controlling the robot via the TCP socket, and display live video from the Kinect. An Android device is capable of establishing a Bluetooth connection to the robot, and send velocity commands over this connection.

Performance and Assessment

Testing sessions, performed in both a simulator and with a live mobile robot prototype, demonstrated RTAB-Map's ability to build maps and localize the robot within these maps. Loop closure detection would work when a sufficient amount of visual features were available, but this was often not the case. Using laser scans as a source of

odometry was susceptible to errors in featureless areas, when the robot rounded corners and when people walked by or towards the robot.

Navigation tests on the live robot demonstrated that the robot can navigate successfully in known and structured environments with some maneuvering space. Navigation performance decreased in cluttered environments, e.g. office environments with many tables placed closely together.

Recommendations

The mobile base requires an overhaul, and a new drive system should be considered. A new drive system should be dimensioned to support the robots weight and be more rugged, so it can handle uneven surfaces. A holonomic drive system could yield a more agile robot.

The Kinect and the LIDAR is currently placed in the front of the base. As the Kinect is unable to reliably measure depth any closer than $0.8m$. A new sensor location should be considered to avoid this blind-zone.

As for RTAB-Map there are factors that speak against the method as a mapping system of an offshore robot. The shortcomings are essentially related to robustness to how the visual appearance of the surrounding change over time. It is recommended to continue testing of RTAB-Map. The tests should include different sensor configurations and stereo cameras, as well as different lighting conditions.

The implementations presented in this thesis is the first attempt at integrating the mobile robot prototype with ROS. It is concluded that ROS is a good tool for prototyping, and it is recommended to continue using the framework in subsequent projects on robotic maintenance. The current software is functional, but not much more. Future projects should strive to increase the system's usability and robustness.

References

[Ama15] Director General Yukiya Amano. *The Fukushima Daiichi Accident, Report By the Director General*. IAEA, 2015. http://www-pub.iaea.org/MTCD/Publications/PDF/Pub1710-ReportByTheDG-Web.pdf.

[and] Bluetoothdevice, android referencel. https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html. Accessed: 04-2016.

[ARG] ARGOS challenge. http://www.argos-challenge.com/en. Accessed: 19-05-2016.

[AS12] David A. Anisi and Charlotte Skourup. A step-wise approach to oil and gas robotics. In *Proceedings of the 2012 IFAC Workshop on Automatic Control in Offshore Oil and Gas Production*, June 2012.

[Asp13] Petter Aspunvik. Robotisert vedlikehold. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2013.

[ATE] ATEX directive: first edition of the atex 2014/34/eu guidelines. http://ec.europa.eu/growth/sectors/mechanical-engineering/atex/. Accessed: 13-05-2016.

[Bek10] Kristian Saxrud Bekken. Bevegelsesstyring av robotarm og kamera med kollisjonsunngåelse. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2010.

[Ber13] Mikael Berg. Navigation with simultaneous localization and mapping. Master's thesis, Dept. of Engineering Cybernetics, NTNU, 2013.

[BMNK13] Kai Berger, Stephan Meister, Rahul Nair, and Daniel Kondermann. *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*, chapter A State of the Art Report on Kinect Sensor Setups in Computer Vision, pages 257–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[Bog16] Robert Bogue. Europe continues to lead the way in the collaborative robot business. *Industrial Robot: An International Journal*, 43(1):6–11, 2016.

[Bro13] Kevin Brothaler. *OpenGL ES 2 for Android*. The Pragmatic Programmers, 2013.

[Coo11] Gerald Cook. *Mobile robots: navigation, control and remote sensing*. John Wiley & Sons, 2011.

[dep12] Sensabot: A safe and cost-effective inspection solution. *Journal of Petroleum Technology*, 64:32–34, 10 2012.

[DRC] DARPA robotics challenge. http://www.theroboticschallenge.org/overview. Accessed: 2016-04-05.

[E24] E24.no: Denne plattformen skal fjernstyres fra land. http://e24.no/energi/statoil/produksjonen-paa-valemon-er-i-gang/23366972. Accessed: 28-05-2016.

[EHS+14] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on*, 30(1):177–187, 2014.

[ER12a] Mohamed A. El-Reedy. Chapter 6 - corrosion protection. In Mohamed A. El-Reedy, editor, *Offshore Structures*, pages 383 – 443. Gulf Professional Publishing, Boston, 2012.

[ER12b] Mohamed A. El-Reedy. Chapter 8 - risk-based inspection technique. In Mohamed A. El-Reedy, editor, *Offshore Structures*, pages 563 – 634. Gulf Professional Publishing, Boston, 2012.

[Foo13] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[GC11] Peter Gorle and Andrew Clive. Positive impact of industrial robots on employment. Report, METRA MARTECH Limited, 2011.

[GP08] Birgit Graf and Kai Pfeiffer. Mobile robotics for offshore automation. In *Proceedings of the EURON/IARP International Workshop on Robotics for Risky Interventions and Surveillance of the Environment, Benicassim, Spain*, 2008.

[hok] URG-04LX-UG01 LIDAR specifications. http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/URG-04LX_UG01_spec_en.pdf.

[HWB+13] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[ifr] International federation of robotics - statistics. http://www.ifr.org/industrial-robots/statistics/. Accessed: 2016-04-05.

[JWA+12] J. Jamieson, L. Wilson, M. Arredondo, K. Evans, J.and Hamilton, and C Sotzing. Autonomous inspection vehicle: A new dimension in life of field operations. *Offshore Technology Conference*, April 2012.

[KFO12] Shinji Kawatsuma, Mineo Fukushima, and Takashi Okada. Emergency response by robots to fukushima daiichi accident: summary and lessons learned. *Industrial Robot: An International Journal*, 39(5):428–435, 2012.

[kina] Kinect for windows, does in work with ros? http://answers.ros.org/question/12876/kinect-for-windows/. Accessed: 10-02-2016.

[kinb] Kinect for windows, hack. http://projects.csail.mit.edu/pr2/wiki/index.php?title=Kinect_for_Windows. Accessed: 10-02-2016.

[kinc] Kinect for windows programming guide. https://msdn.microsoft.com/en-us/library/hh855348.aspx.

[kind] Kinect for windows, specifications. https://msdn.microsoft.com/en-us/library/jj131033.aspx. Accessed: 10-06-2016.

[Kine] Kintinous. https://github.com/mp3guy/Kintinuous. Accessed: 2016-03-21.

[KLT09] Erik Kyrkjebø, Pål Liljebäck, and Aksel A. Transeth. A robotic concept for remote inspection and maintenance on oil platforms. In *Ocean, Offshore and Arctic Engineering, ASME 2009 28th International Conference on*, 2009.

[KMP15] K. Kydd, S. Macrez, and P. Pourcel. *Autonomous Robot for Gas and Oil Sites*. Society of Petroleum Engineers, September 2015.

[KMvSK11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[Kou16] Anis Koubaa. *Robot Operating System (ROS): The Complete Reference*, volume 1. Springer, 2016.

[Lin15] Vegard Stjerna Lindrup. Visual sensing in mobile robots. Proj. rep., Dept. of Engineering Cybernetics, NTNU, 2015. 9th semester specialization project report.

[LM13] M. Labbe and F. Michaud. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[LM14] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[MEBF+10] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 2010.

[MIM] MIMROex, mobile maintenance and inspection robot for process plants. http://www.ipa.fraunhofer.de/fileadmin/user_upload/Kompetenzen/Roboter-_und_Assistenzsysteme/Industrielle_und_gewerbliche_Servicerobotik/English_Documents/Product_sheet_MIMROex_Mobile_maintenance_and_inspection_robot_for_process_plants.pdf.

[NIH+11] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[PBB11] K. Pfeiffer, M. Bengel, and A. Bubeck. Offshore robotics - survey, implementation, outlook. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 241–246, Sept 2011.

[pre] Press release: Subsea 7 completes design and build of first commercial autonomous inspection vehicle (aiv). http://www.subsea7.com/content/dam/subsea7/Company%20News/2011/Subsea7completesdesignandbuildoffirstcommericalAIV.pdf. Accessed: 2016-04-13.

[PSM+16] Ian Peerless, Adam Serblowski, Berry Mulder, et al. A robot that removes operators from extreme environments. In *SPE International Conference and Exhibition on Health, Safety, Security, Environment, and Social Responsibility*. Society of Petroleum Engineers, 2016.

[QGS15] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS*. O'Reilly Media, Inc., December 2015.

[RCR+15] Pere Ridao, Marc Carreras, David Ribas, Pedro J. Sanz, and Gabriel Oliver. Intervention auvs: The next challenge. *Annual Reviews in Control*, 40:227 – 241, 2015.

[ROSa] catkin workspace wiki page. http://wiki.ros.org/catkin/workspaces. Accessed: 2016-02-05.

[ROSb] ROS create package tutorial. http://wiki.ros.org/ROS/Tutorials/CreatingPackage. Accessed: 2016-02-05.

[ROSc] ROS history. http://www.ros.org/history/. Accessed: 2016-02-28.

[ROSd] ROS-industrial. http://rosindustrial.org/the-challenge/. Accessed: 2016-04-05.

[ROSe] ROS-industrial, supported hardware. http://wiki.ros.org/Industrial/supported_hardware. Accessed: 2016-04-05.

[ROSf] ROS installation. http://wiki.ros.org/indigo/Installation/Ubuntu. Accessed: 2016-02-29.

[ROSg] ROS navigation tuning guide. http://wiki.ros.org/navigation/Tutorials/Navigation%20Tuning%20Guide.

[ROSh] ROS on the iss. http://www.ros.org/news/2014/09/ros-running-on-iss.html. Accessed: 2016-04-10.

[ROSi] roslaunch. http://wiki.ros.org/roslaunch. Accessed: 2016-03-15.

[rta] Setup rtab-map on your robot! http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot.

[sta] Offshore.no: Statoil vil bygge flere folkefrie plattformer. http://offshore.no/sak/63114_statoil_vil_bygge_flere_folkefrie_plattformer. Accessed: 28-05-2016.

[sub] Teknisk ukeblad: Ubemannede plattformer skal konkurrere med subsea. http://www.tu.no/artikler/ubemannede-plattformer-skal-konkurrere-med-subsea/226868. Accessed: 28-05-2016.

[tcp] Linuxhowtos.org - sockets tutorial. http://www.linuxhowtos.org/C_C++/socket.htm. Accessed: 03-2016.

[TSSO+10] Aksel A Transeth, Øystein Skotheim, Henrik Schumann-Olsen, Gorm Johansen, Jens Thielemann, and Erik Kyrkjebø. A robotic concept for remote maintenance operations: A robust 3d object detection and pose estimation method and a novel robot tool. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5099–5106. IEEE, 2010.

[Vin14] Jan-Erik Vinnem. *Offshore Risk Assessment vol 1.: Principles, Modelling and Applications of QRA Studies*. Springer London, London, 2014.

[WA12] Jarrett Webb and James Ashley. *Beginning Kinect Programming with the Microsoft Kinect SDK*. Apress, 2012.

Appendix A

Setting Up the Project

A.1 Hardware Setup

For a guide on how to connect the hardware, consult figure 4.4 and 4.2. Otherwise, see the hardware list. A principal connection diagram for the network is shown in figure A.1. The router should already be configured for communications. If not, consult [Asp13].

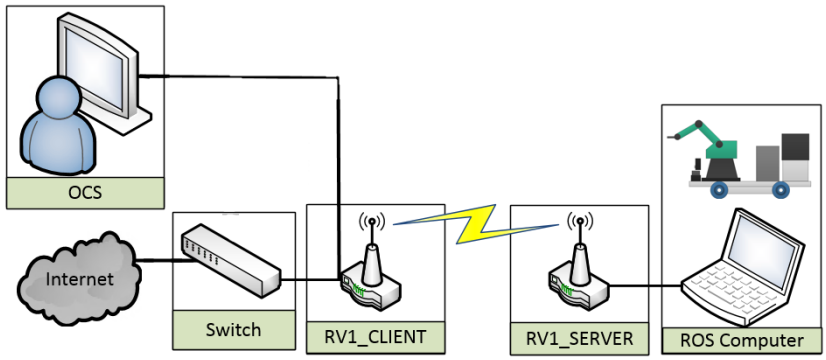


Figure A.1: Network hardware setup.

Lead Battery Safety Precautions

The lead battery used in this project (black 24V 48Ah, Biltema), contains a large amount of energy and highly corrosive sulfuric acid. Remember to read the warning label on the battery, and handle the battery with care.

An explosive mix of hydrogen and oxygen may be formed when charging the battery. Some safety precautions are necessary to handle the risk:

- Always charge the battery in a well ventilated area.

- Turn the charger off before removing the connecting clamps.
- Allow some time to pass after charging before connecting or disconnecting cables to the battery poles.

Equipment List

- A computer that can be mounted on the robot. The computer must run on Linux Ubuntu (13.10 or 14.04 for ROS Indigo) and have a Bluetooth adapter.
- Two wireless routers (for example TP-Link).
- Two sinus inverters. One power inverter from Biltema and a silver colored pure sine inverter.
- One 12V Battery. The battery used in this project has a capacity of $45Ah$.
- A $230VAC/24VDC$ converter.
- One XMEGA A3BU evalation board.
- One Hokuyo URG-04LX-UG01 LIDAR.
- A Kinect for XBOX 360 or an equivalent OpenNI depth camera.

A.2 Installation

A.2.1 Software list

This guide assumes that ROS Indigo is being used on a comparable system, for example Ubuntu 14.04. A guide for installing either ROS or Ubuntu is best aquired elsewhere. For ROS, see ros.org. Indigo was chosen primarily because of `openni_launch`.

Hector SLAM for ROS Install with

```
sudo apt-get install ros-indigo-hector-slam
```

Web video server node Install with

```
sudo apt-get install ros-indigo-web-video-server
```

LIDAR driver Install with

```
sudo apt-get install ros-indigo-hokuyo-node
```

RTAB-Map Installed from source. Guide available at
https://github.com/introlab/rtabmap/wiki/Installation#ros

Gazebo Install with

```
sudo apt-get install ros-indigo-gazebo-ros-pkgs
```

Otherwise, consult http://wiki.ros.org/gazebo_ros_pkgs.

A.3 Configuring the Project

A.3.1 Configuring the ROS Workspace

Assuming that ROS is installed and properly configured, the first step in configuring the project is to create a catkin workspace. The custom ROS packages bundled with the digital attachments, should be placed in the `src` folder within the catkin workspace.

A.3.2 Configuring the Bluetooth Connection

The Qt framework is used to simplify the implementation of the Bluetooth connection between the ROS graph and a remote device. Our ROS installation for this project already includes some variant of Qt version 4.8. While useful for creating new GUI applications, it lacks a Bluetooth API. The latest version of Qt, version 5.x, is equipped with libraries necessary for developing Bluetooth applications. This part of the guide will explain how to create a Qt 5 application which can be build by `catkin_make` and run as a *rosnode*.

1 - Install Qt5

Installing Qt5 for Linux is a straight forward procedure. Go to qt.io, and download the free version of Qt. All necessary instructions are provided. Qt5 may be installed in the home folder.

2- Enabling Qt5 in a ROS node

It is assumed that the ROS package `bluetooth_server`, is located in a catkin workspace:

```
<NAME OF CATKIN WORKSPACE>/src/bluetooth_server
```

Inside this folder, open the file ”CMakeLists.txt” and locate the following:

```
set(CMAKE_PREFIX_PATH "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5"
  "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5Core"
  "/home/vegard/Qt/5.5/gcc_64/lib/cmake/Qt5Bluetooth"
```

Change these paths to the correct paths on your system.

A.4 System Launch Procedure

This procedure assumes that the ROS implementation source code is placed in the `src` folder of a catkin workspace, and that he project has been built successfully. The first step is to go through a hardware checklist:

1. Verify that the motor control board is connected to the wheel drivers, as shown in figure 4.14.
2. Kinect, router and motors are connected to either internal or external power.
3. Ensure that all USB ports are free. Exceptions apply to devices such as the Kinect or a mouse.
4. Connect the Hokuyo lidar (URG-04LX-UG01) to a USB port.
5. Connect the motor control card to a USB port. Ensure that it is connected after the LIDAR. Ensure that the correct firmware is installed on the board.

This procedure is recorded to video ”start_live_robot”:

1. Open five terminal windows.
2. Launch `roscore` in one of the windows
3. `cd` to `<your_catkin_workspace>/src/mar/scripts>`
4. In this folder, run `$ sudo ./setup_hokuyo.sh`
5. In the remaining three terminals, `cd` to your catkin workspace.
6. In each of the terminals, run `$ source ./devel/setup.bash`
7. In one terminal, bring up the robot with
   `$ roslaunch mar mar_bringup.launch`
8. In another terminal, launch rtabmap with
   `$ roslaunch mar rtabmap.launch`
   Consult the launch file to see argument options.

9. In the last terminal, launch navigation with
   `$ roslaunch mar_2dnav move_base.launch`

Appendix B
Robot Mass Calculations

Mass density, Aluminium: 2,7 g/cm^3

----- Plate: -----

Volume: 37x80x0,5 cm^3 = 1480 cm^3
Mass: 1480 cm^3 x 2,7 g/cm^3 = 4 kg

----- Bottom: -----

Volume: Four side plates: 2 x 40 x 5 x 0,2 cm^3 = 160 cm^3
Aft and front: 2 x 36 x 5 x 0,2 cm^3 = 72 cm^3
Miscellanious : 4*70 = 280 cm^3

Total: 440cm^3

Mass: Aluminium: 440 cm^3 * 2,7g/cm^3 = 1188 g = 1,2 kg

Wheel: reference:
http://www.superdroidrobots.com/shop/item.aspx/omni-wheel-and-shaft-assembly-double-row/383/

Mass m. shaft: 1,8 lbs = 816 g

Total mass, wheels: 4*816g = 3264 g = 3,3 kg

```
----- Arm: -----

Robot arm: reference: http://www.intelitek.com/robots/scorbot-er-4u/
Controller reference: http://www.intelitek.com/robots/usb-controller/

Robot arm mass: 10,8 kg
Robot controller mass: 7 kg

----- Rear compartment (rack): -----

Guessing at 20 kg (battery, computers, chassis etc.)
l_x = 35 cm
l_y = 37 cm
l_z = 40 cm
```

Appendix C
Troubleshooting

C.1 Introduction

This chapter contains proposed solutions to some of the problems that was encountered over the course of the semester. The solutions are not complete or comprehensive, but may provide some quick fixes for any students that may continue working with this project.

C.2 Hardware

The Wheel Fell Off!

During a test drive with the robot, the base collapsed because one of the wheel shafts had slipped out of the motor drive shaft. The solution to the problem is simply to tighten the set screw which connects the motor shaft to the wheel. The set screw is shown in figure C.1.

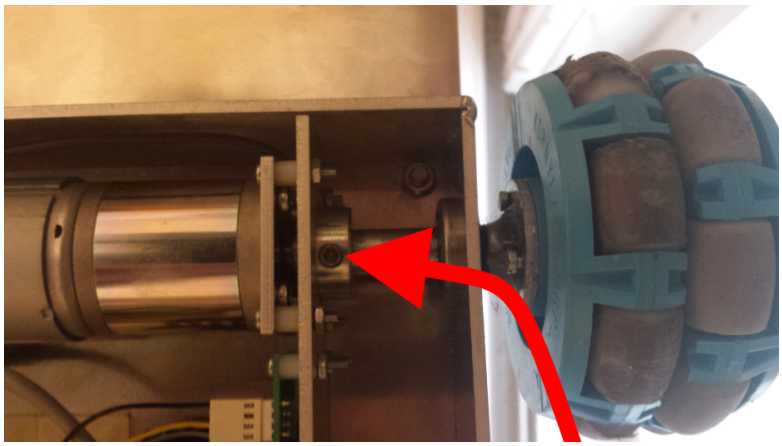


Figure C.1: The set screw which holds the wheel onto the motor drive shaft.

C.3 ROS

ERROR: tf2:ExtrapolationException

When running the released ROS distribution binary of `RTAB-Map` installed with `apt-get` on the robot, the node would crash after a few iterations. The error message is as follows:

```
Lookup would require extrapolation into the future, ..., when looking
up transform from frame [laser] to frame [base_link].
```

The requested transform is milliseconds ahead of "now". This issue was fixed by maintainers in March 2016[1], but was not yet integrated into the released binary. During work with this project, the problem was solved by building `RTAB-Map` from source, where the most recent fixes are included. This is a straight forward procedure, which is described on the project's GitHub repository.

C.4 Gazebo

Error [Node.cc.90] No namespace found

Solution: Remember to source the *gazebo* installation. In this case, with *gazebo-2.2* installed as recommended for ROS Indigo, the setup file can be sourced by typing

```
$ source /usr/share/gazebo-2.2/setup.sh
```

Dependency Issues When Installing *gazebo2*

This problem was encountered after removing gazebo and then typing

```
$ sudo apt-get upgrade
```

When typing

```
$ sudo apt-get install -y gazebo2
```

the installation failed because some dependencies had been upgraded to an incompatible version. To solve this, take note of the missing dependencies listed after entering

[1] https://github.com/introlab/rtabmap_ros/issues/54

the command above, open Ubuntu Software Center and select the History tab. Scroll down and locate the missing dependencies. They should have a red X next to them, indicating that they have been uninstalled. Then, enter the following command:

```
$ sudo apt-get install <NAME OF THE UNINSTALLED DEPENDENCY>
```

C.5 Ubuntu

Ubuntu Freezes

Sometimes during work with the project, Ubuntu would freeze and become unresponsive to keyboard input and mouse clicks. The mouse could be moved around, but was otherwise unresponsive. This event occurred exclusively when using `rviz` and displaying a camera topic as an image in the lower left corner of the GUI. The following steps from a post at askubuntu.com[2], solves the problem.

While holding Alt and SysReq (Print Screen), type R E I S U B. Press each key properly, and allow a few seconds to pass between each keystroke so that each command has time to execute. This should cause the computer to reboot, and is supposedly safer than using the power button. See the footnote for more information.

[2]What to do when Ubuntu freezes: http://askubuntu.com/questions/4408/what-should-i-do-when-ubuntu-freezes/36717#36717

Appendix D
DVD Contents

- Master Thesis
- Project files: Robot (ROS)
- Project files: Operator Control Station (Qt)
- Project files: ”Robot Leash” (Android)
- Project files: ”mar_motor_driver” (C - Atmel)
- Videos
  - live_3d_obstructions
  - live_first_large_scale_mapping
  - live_mapping_odom_drift
  - live_mapping_succesful
  - live_multi_session_mapping
  - live_navigation_1
  - live_navigation_2
  - Obstruction detection and avoidance
  - ocs_test
  - ocs_test2
  - sim_global_plan
  - sim_global_plan_poor_localization
  - sim_wrong_loop_closure
  - start_live_robot
- Images